AN ENHANCEMENT OF THE COMPUTER TYPESETTING CAPABILITY OF UNIX

Boyd Scott McCord

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

An Enhancement of the
Computer Typesetting Capability of UNIX

by

Boyd Scott McCord

June 1977

Thesis Advisor: G. L. Barksdale, Jr.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Enhancement of the Computer Typesetting Capability of UNIX		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1977
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Boyd Scott McCord		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1977
		13. NUMBER OF PAGES 139
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer typesetting, digitized fonts, text processing, typeface, fonts		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Variable width fonts were adapted for use in a 16-bit environment, and an existing font editor was modified to provide for the creation and maintenance of such fonts. A UNIX compatible file format was designed for the storage of digitized characters, and a set of font-manipulation programs were written. These developments enhance the digital typesetting capability of UNIX. Thirty-four fonts

were obtained from the Stanford Artificial Intelligence Laboratory (SAIL) and were modified for use under UNIX. The fonts offer a variety of sizes and styles; their selective use allows for a more compact and aesthetic display of textual information in documents produced on a Versatec 1200-A printer/plotter.

An Enhancement of the Computer Typesetting Capability of UNIX

by

Boyd Scott McCord
Captain, United States Marine Corps
B.S., United States Naval Academy, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1977

Thesis
M1813
c.1

ABSTRACT

Variable width fonts were adapted for use in a 16-bit environment, and an existing font editor was modified to provide for the creation and maintenance of such fonts. A UNIX compatible file format was designed for the storage of digitized characters, and a set of font-manipulation programs were written. These developments enhance the digital typesetting capability of UNIX. Thirty-four fonts were obtained from the Stanford Artificial Intelligence Laboratory (SAIL) and were modified for use under UNIX. The fonts offer a variety of sizes and styles; their selective use allows for a more compact and aesthetic display of textual information in documents produced on a Versatec 1200-A printer/plotter.

CONTENTS

I. INTRODUCTION

A. BACKGROUND

1. Technological Progress

The invention of moveable type had a tremendous impact on man's environment. The invention of printing, more than any single achievement, "marks the line of division between medieval and modern technology" [Ref. 15]. Technological improvements to Gutenberg's invention continued at a snail's pace through the latter half of the nineteenth century until the public demand for daily news required more rapid modernization in the printing industries. Constant improvements, one of which was the use of electrical power to drive presses, continued through the mid twentieth century until, finally, one could find printing rooms filled with linecasters, complex electro-mechanical contraptions producing hot metal type, and presses noisily spewing forth tons of paper each day. However, the public's appetite was seemingly insatiable, and, while press speed was satisfactory, the entire process was slowed by the composition functions of line justification, hyphenation, spelling, and so on. These functions still required human preprocessing or operator interruption of linecasters. The solution to this bottleneck was provided by the computer and recent advances in text processing.

2. Computer Assisted Typesetting

The computer, gradually extending its influence into many unrelated fields, had now entered the printing industry. Its employment was in assisting the typesetters, not replacing them. Computer assisted typesetting (CATS) was characterized by the computer performing all line justification, page breaking, hyphenation, etc.; essentially the information to be printed was being preprocessed by a program. The result of this process was a tape, either a perforated paper tape or a magnetic tape, which contained the processed text interspersed with commands to the printing device, specifing when to hyphenate, when to change typeface, when to indent, etc. In addition to processing text, the computer's software had to be tailored to the specific printing apparatus. The tape was designed to assist the electro-mechanical linecasters in the setting of hot metal type, and presses continued to produce the print.

Soon many newspapers and publishers were using computers to produce tapes of processed text, which drove the more conventional linecasting machines. Although the future of computers in the printing industries looked bright, there were repercussions and even some failures. Labor revolted at the smell of further automation, and this problem was necessarily handled delicately. The WASHINGTON EVENING STAR was the first major newspaper to successfully assist its press operation with a computer [Ref. 6]. One such attempt failed. In 1962, the ARIZONA JOURNAL was founded, and its

publisher decided to begin with a computer, a GE 225, to perform text processing functions and administrative tasks. GE's computer personnel working on the project knew nothing of publishing newspapers, and the paper's staff knew nothing of computers. The software did not materialize in time to begin printing, and creditors foreclosed [Ref. 16].

3. Computer Typesetting

In the early 1960's, there was a flurry of development in nonimpact printing. Previously, all printing was by impact, the striking of a raised shape of a character onto paper with some inking mechanism involved. Using new advances in xerography, photography, and high speed control mechanisms, nonimpact printing devices were characterized by higher printing speeds, less noise, fewer moving parts, higher reliability, and a greater capacity to handle both textual and graphical information [Ref. 4].

In 1961 Micheal P. Barnett, at MIT, designed a program which processed text and produced tape which drove a Photon 560, a phototypesetter. This particular phototypesetter contained a disc with photographic images of characters from various typefaces. The typefaces were arranged in concentric circles about the center of the disc. As the tape was processed, a light source moved back and forth from the center of the disc to its edge while the disc rotated at high speed. An intricate timing mechanism within the Photon 560 ensured that pictures of the correct charac-

ters in the correct typeface were exposed to the film behind the disc. From the film, either lithographic plates could be made or documents produced directly through special machines [Ref. 1]. Distinguishing between computer typesetting and computer assisted typesetting (CATS) is difficult. Considering the tremendous potential of nonimpact printers and the recent (last 10 years) advances in computer output microfilm (COM), computer typesetting is, in this author's opinion, the future direction in the printing industry. Computer typesetting tends to be more software oriented. Consequently, there tends to be less of a separation between the text processing and the actual character generation; continuity is more apparent in computer typesetting. In computer typesetting, the computer sets "software" type; whereas, in CATS, the computer creates some tape which drives devices which set "hardware" type. The state-of-the-art character generation techniques for computer typesetters are photo/optic, photo/scan, and digital/scan [Refs. 2 and 14]. Briefly, the three techniques are described:

a. Photo/optic

Here, photographic images of characters are stored, and high speed access to these images allows them to be projected through a lens onto film or paper. Scaling is possible through lens switching, and access times are several milliseconds per character. The presence of moving parts limits speed and reliability.

b. Photo/scan

Again, photographic images of characters are stored. The selected character image is "scanned"; that is, horizontal slices of its image are projected sequentially from top to bottom completing the full character picture. Scaling is possible by expanding and/or adding duplicate scan lines. Again, the presence of moving parts limits reliability.

c. Digital/scan

All character images are stored in memory as pictures composed of "1's" and "0's" (bits either "on" or "off"). The character images are plotted by a program passing the digital picture, in bits, to static printing heads (one head per bit) or by recording the digital picture, bit by bit, with an electron or laser gun. Although scaling is not possible, this technique provides the fastest character access and interfaces with printing devices with few moving parts. These last two characteristics are important advantages. One disadvantage, however, is that digital representations show a "staircase" effect in large characters.

Producing different typefaces can be accomplished for each technique. The photo/optic and photo/scan methods require that a master disc of character images be made. Digital/scan devices acquire different typefaces from either of two ways. First, there are devices that can "read" a photographic character image and pass a digitized

interpretation to memory for storage. Secondly, there exist interactive programs (editors) which enable a user to create digitized character pictures [Ref. 2]. There was no attempt to analytically compare the three techniques or to propose benchmarking methods as this thesis effort was restricted to a system which is hardware dependent on the digital/scan technique.

Although there is much ongoing research in the area of computer typesetting, its application is well established in the printing industry. For example, as early as 1970, a book was published entirely by computer typesetting [Ref. 3].

B. COMPUTER TYPESETTING UNDER UNIX

1. System Design

This section describes the system as it existed before this author began his research effort, and, although many improvements were made, not all system components have been modified; however, the system may still be utilized in its original configuration if desired. The basic components of the system are described below:

a. Troff

Troff is a text processor similar to Nroff, and both were designed at the Bell Laboratories by the same author [Refs. 8 and 9]. Troff, however, accepts additional

commands to change typefaces (hereafter referred to, and later defined, as "fonts"). Normally, the output from Troff goes directly to a phototypesetter. There being no such device at NPS, Troff has been modified, and its output becomes an intermediate file which is processed by Vts.

b. Vts

Vts is a virtual typesetter, a program which takes the preprocessed text lines from the file from Troff and which then sets the required digitized character definitions into plot buffers. The plot buffers are sent to a Versatec printer/plotter.

c. Edf

Edf is a font editor, an interactive program that enables the user to create and maintain digitized fonts. In its original form it processed only fixed width fonts of a specific size.

d. Font Library

The original font library consisted of four fonts. Three contained the standard ASCII character set, and the fourth contained special characters for setting mathematical formulas.

e. Display

There is a display program which will plot, on the Versatec, all characters in a font.

The system components described above were designed and programmed by Professor G.L. Barksdale. They are intended to manipulate fixed width fonts, the characters of which are all 16 pixels wide and 20 pixels high. A "pixel" is a unit of resolution (a picture element) on a plotter. On the Versatec, there are approximately 200 pixels per inch. In setting digitized characters, a plot buffer represents one horizontal line (raster line) of pixels, i.e., 20 plot buffers would need to be sent to the Versatec to plot ("print") a line of text. Figure 1 illustrates the interplay among the various system tools.

SYSTEM DESIGN

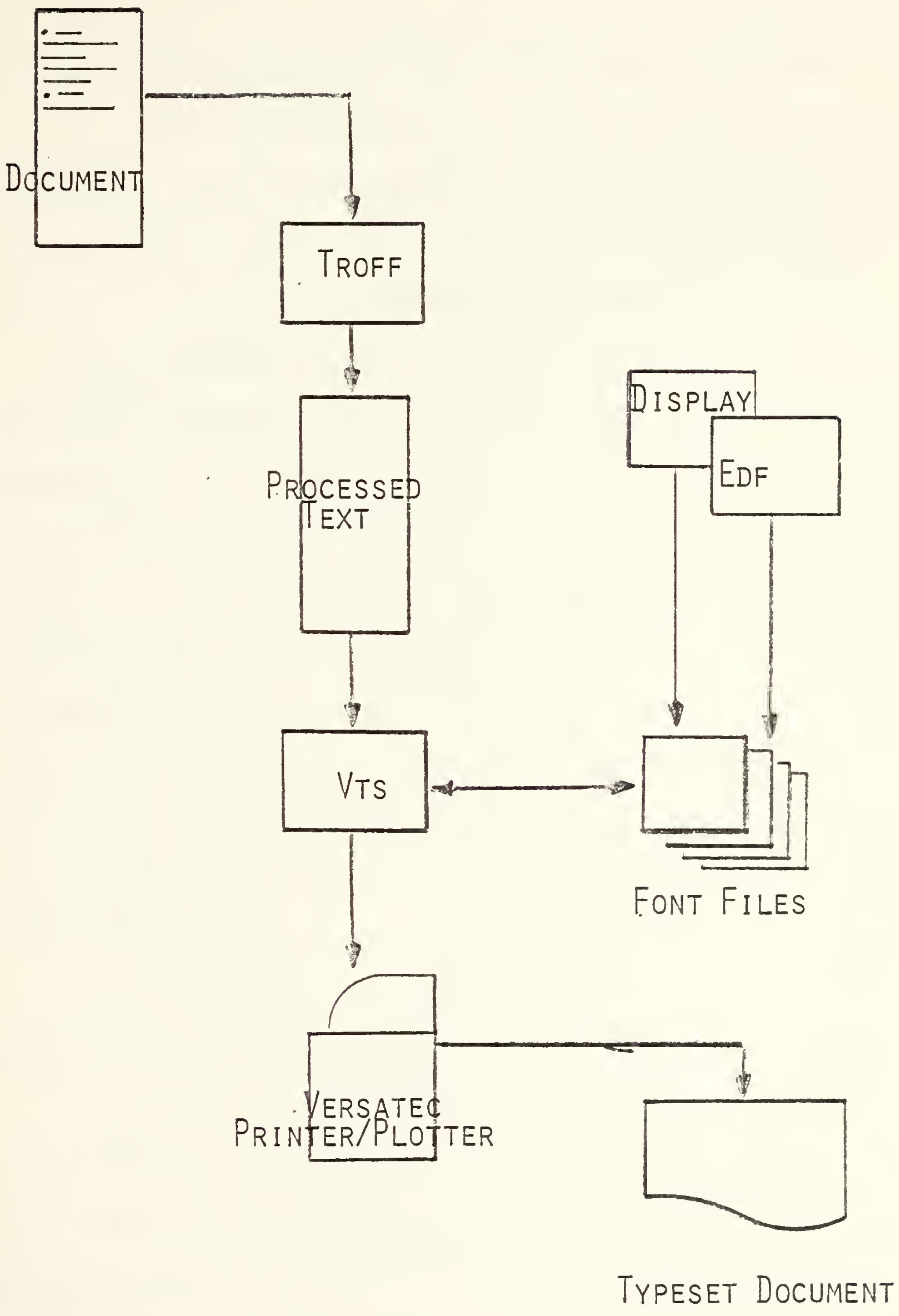


Figure 1

2. Enhancement Objectives

The original objective of the thesis research was to increase the font library to include variable width fonts of various sizes. These fonts, 34 in all, were obtained in machine form from the Stanford Artificial Intelligence Laboratory (SAIL). Additional objectives were to modify the system components to handle the variable width fonts, and to add to Vts a limited plot (simultaneous text/graphics) capability. In conjunction with this author's thesis, LT. P.M. Doyle adapted the Hershey Character Sets for use in graphics and typesetting. LT. Doyle developed a program which converted the vector formatted Hershey font files to digitized font files; a scaling option was made available in the conversion program. The modifications to the system tools were the results of both theses [Ref. 5]. The following sequence was designed to attain the thesis objectives:

a. Design a UNIX compatible file format for digitized font storage.

b. Convert fonts from SAIL to NPS file format, correcting any detected errors.

c. Modify Edf to handle variable width fonts.

d. Modify the system font display program to display variable width fonts.

e. Modify Troff.

f. Modify Vts.

g. Document the new system (write a user's manual).

h. Document the program development and the thesis research (write the thesis).

The objectives enumerated in a. through d. were completed; however, their completion required more time then was anticipated (primarily due to the debugging and testing phases of program development). Therefore, objectives e. and f. were omitted. In their places, a program called Signmkr was written. Signmkr sets type in the same manner that Vts would have, had it been modified. Additionally, its design included the capability of some simple text processing functions. Objectives g. and h. were completed. The user's manual was published separately as an Technical Note [Ref. 7] and received distribution as such. The remainder of this thesis documents the objectives meet and is concerned primarily with program design and development. The final chapter presents conclusions concerning the resulting system configuration for computer typesetting under UNIX and some ideas for future developments in this field.

II. SAIL FONTS

A. DESCRIPTION

1. Origin

A font is a collection of character images, all of which are of the same style and height, which are mapped into some character set. Fonts are in general freely exchanged among academic institutions, primarily through ARPA. The SAIL fonts, named for the agency from which acquired, were obtained in digitized, machine-readable form on magnetic tape [Ref. 12]. There were 34 fonts in all, and they are catagorized as follows:

a. Bodoni and Nonie fonts. These two groups of fonts each have distinct designs and each contain variable width fonts of different sizes and styles. Together, they account for 23 of the 34 fonts.

b. GRFX (Graphics). There are two fixed width fonts which provide a limited graphics capability. They are useful for setting flowcharts, tree structures, and simple graphs. They are also the only two fonts in which "kerning" occurs.

c. Math. There are five fonts that contain special mathematical symbols for setting formulas.

d. SAIL10. This font is the only text oriented fixed width font in the library.

e. SIGNS. There are three fonts which are large and excellent for entitling documents and making signs: SHD15, SIGN22, SIGN41.

The terms used to describe fonts, e.g., variable width, kerning, etc., and the meanings of font and character dimensions are discussed in the next section. Some of the acquired fonts were originally designed at MIT, others at CMU (Carnegie-Mellon), and the remainder at Stanford. Stanford generally names, or has renamed, all fonts so that the trailing character or numbers connote size in pixels. The scheme for naming fonts at NPS is similar but denotes size in points, the traditional printer's measure. The 34 fonts added to the library are listed in Table 1.

SAIL FONTS

8 point Bodoni Mathematical	BDJ8
10 point Bodoni	BDR10
10 point Bodoni Italic	BDI10
10 point Bodoni Mathematical	BDJ10
10 point Bodoni Bold	BDR10X
12 point Bodoni	BDR12
12 point Bodoni Italic	BDI12
12 point Bodoni Bold	BDB12
15 point Bodoni	BDR15
15 point Bodoni Italic	BDI15
25 point Bodoni	BDR25
10 point Nonie	NONS
10 point Nonie Italic	NONSI
10 point Nonie Bold	NONSB
10 point Nonie Bold Italic	NONSBI
12 point Nonie	NONM
12 point Nonie Italic	NONMI
12 point Nonie Bold	NONMB
12 point Nonie Bold Italic	NONMBI
14 point Nonie	NONL
14 point Nonie Italic	NONLI
14 point Nonie Bold	NONLB
14 point Nonie Bold Italic	NONLBI
10 point Graphics	GRFX10
14 point Graphics	GRFX14
10 point Math	MATH10
13 point Math	MATH13
15 point Math	MATH15
20 point Math	MATH20
21 point Math	MATH21
10 point Delegate	SAIL10
15 point Shadow	SHD15
22 point Sign	SIGN22
41 point Sign	SIGN41

Table 1

2. Font / Character Dimensions

In order to manipulate fonts and the characters within them, there are attributes of fonts and their characters which provide information to typesetting programs. These attributes are the dimensions and accessing information.

a. Font Dimensions

A font is characterized by its height and its logical height, the two most significant dimensions. A third dimension, the width of the widest character in the font, is of less importance. The character picture of each character in the font is conceptually set in a rectangular frame which is as high as the font's height and as wide as the character's raster width. The logical height is the distance from the top of this conceptual frame to the baseline, the imaginary line on which the characters sit. For example, "ascenders", such as an "h", "l", or "t", sit on the baseline; whereas, "descenders", such as a "p" or "q", may extend below it. Two fonts are incompatible if either their heights or their logical heights differ.

Font and character dimensions are measured in pixels which, once again, are units of resolution. On the Versatec printer/plotter, there are 200 pixels per inch. There is another descriptor of font height, the "point". At 200 pixels per inch, one point is approximately 2.8 pixels or about 1/72 inch. Fonts are generally referred to as a

"10 point font", an "8 point font", and so on. Point size is a general size descriptor, pixel height being more exact. For instance, BDR10 is a 10 point font which is 26 pixels high. NONS, another 10 point font, is only 25 pixels high.

Fonts are either fixed or variable width. Being a fixed width font implies that all characters within the font have the same width. Being variable width implies otherwise. Variances in character widths are a significant thorn in the text processing/computer typesetting interface. The text processor requires character widths to perform line justification. Table 2 is an analysis of the character widths for "W's" from various families of both fixed and variable width fonts. An inspection of the table shows the lack of any consistent relationship between font height and character width for fonts in general. As a rule then, Troff cannot compute a character width from the font height. However, by examining the error percent based on a 10 point reference within specific families of fonts, there appear to exist useable relationships within each family. By incorporating tables within Troff for each of the various families, character widths could be computed. The specifications of such a scheme were not fully investigated, but the method appears to be a desirable alternative to the accessing of font files by Troff for the character widths needed for line justification. Finally, fonts may be of different styes. NONS is an "upright" font; NONSI, an italicized version; and NONSB, a bold (heavier) version.

Font Height-Character Width Analysis

FONT	HT(pt)	CW(act)	CW(comp)	%ERROR
NONS	10	23	23	0.00000
NONM	12	27	27.6	2.22222
NONL	14	33	32.2	2.42424
NONSI	10	21	21	0.00000
NONMI	12	26	25.1	3.46135
NONLI	14	32	29.3	8.43721
NONSB	10	23	23	0.00000
NONMB	12	29	27.6	4.82763
NONLB	14	34	32.2	5.29442
BDR10	10	25	25	0.00000
BDR12	12	27	30	11.1111
BDR15	15	38	37.5	1.31516
BDR25	25	63	62.5	0.79342

Table 2

b. Character Dimensions

Figure 2 illustrates font and character dimensions. The raster width is the width of the character picture and is used in accessing the stored character picture. The character width is the space (in pixels) the text processor will allocate a particular character on a line, and it may differ from the raster width. Character width and raster width differ only when kerning occurs. Kerning is a technique whereby characters may be set more closely to minimize white space. In a font that allows kerning, the left and right kerns define how close adjacent characters may be set.

FONT/CHARACTER DIMENSIONS

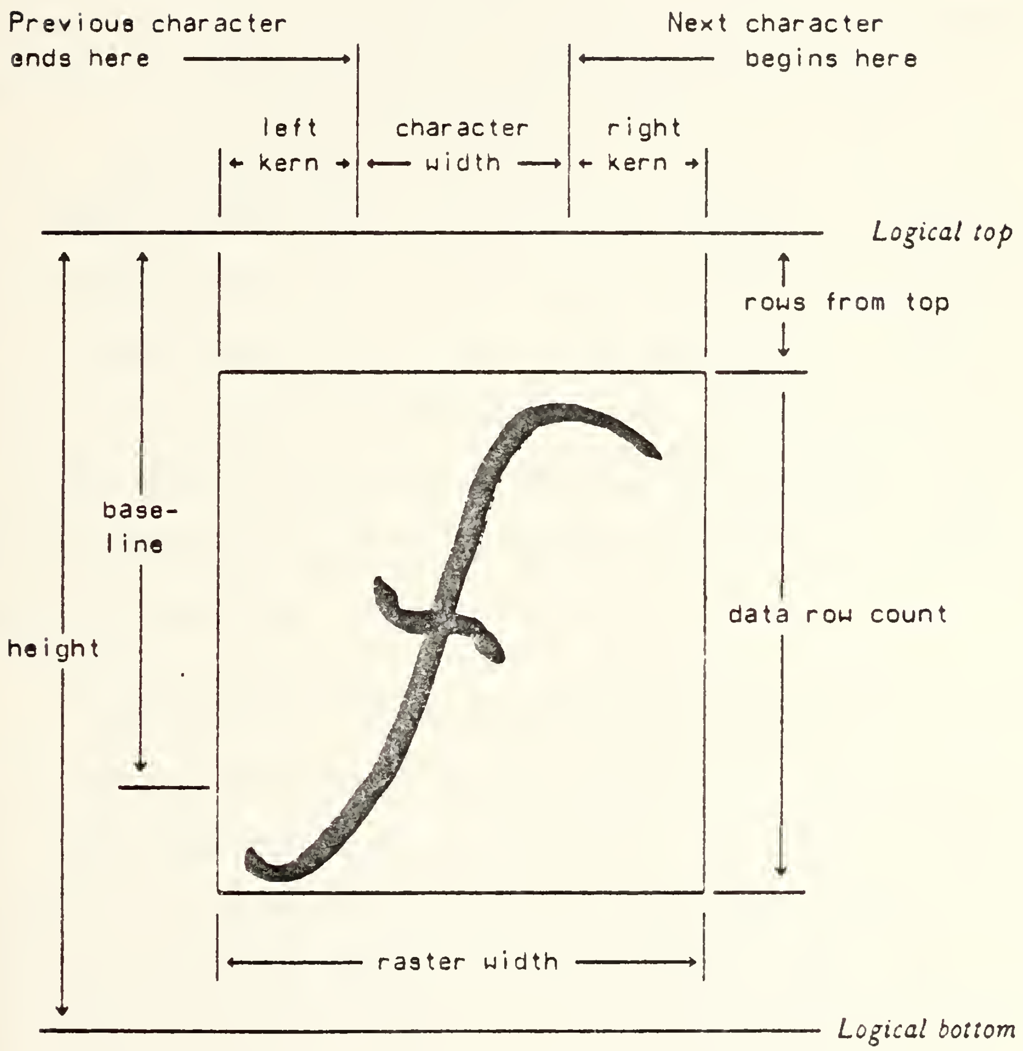


Figure 2

In setting characters with kerning, the bit pictures within the kerns of adjacent characters must be "anded". If the result of the "and" is clear (all zeros), then no character picture overlap will occur, and the kerning is permitted. Otherwise, to prevent the overlap, kerning is aborted, and the character spacing must be determined by raster width. Decisions on kerning are made whenever either of two adjacent characters have coincident nonzero kerns, i.e., either the right kern of the left character is nonzero or the left kern of the right character is nonzero (or both). The capability of setting fonts with kerning was neither a property of the original system nor was it an enhancement objective; however, to facilitate its future implementation, kerning information is stored in character definitions and can be updated by the font editor, Edf. The two fonts where kerning occurs, GRFX10 and GRFX14, require no special treatment by typesetting programs.

Additional character dimensions are solely for accessing the bit picture of the character. The rows-from-top (rft) describes the number of blank (all zero) raster lines above the character picture in the frame; thus, this part of the character picture, which is blank, need not be stored. Data-row-count (drc) is a count of the number of raster lines which form the visible picture and which are stored. Blank raster lines required to fill out the bottom of the picture frame are not stored, and the number of blank lines needed is computed using the font height, rft, and

drc. Figure 3 illustrates that portion of the picture which is stored and the full picture which is expanded by a program.

3. SAIL Font File Format

The SAIL fonts were received as digitized files written on a tape by a PDP-10 at Stanford. The PDP-10 has a 36-bit word with four, 9-bit bytes per word; therefore, reading files from the tape into a PDP-11 file did not leave the information in a readily useable format. For each word of data from the PDP-10, six 8-bit bytes on the PDP-11 were required, the two high order bits of each byte being wasted. Conversion to a more useable, compact font file format was mandatory. The SAIL and NPS font file formats are similar by design; however, a few minor changes have resulted in significant memory savings. Basically, a SAIL font file is broken into three sections:

a. Header Table

At the beginning of the file is a header table. The character code collating sequence is the indexing mechanism for the table, and the table provides random character definition accessing, an absolute necessity when minimizing execution times for setting type. The table contains 128 words, the left half of each word holding the character width and the right half being a pointer to the character definition. A zero character width in any position implies that the particular character is not defined in

the font.

b. Font Dimensions/Description

The font dimensions follow the header table: the font height, the maximum character width, and the font logical height. Immediately following the dimensions is an optional ASCII description of up to 480 characters. Five characters are packed into each 36-bit word, and the description is terminated by an all zero byte ('\0').

c. Character Definitions

The remainder of the file contains the character definitions pointed to by the header table. Each definition follows an identical format and contains character dimensions, the bit picture, and the picture accessing information.

Figures 4 and 5 illustrate, respectively, the SAIL file and character definition formats.

PICTURE STORAGE/EXPANSION

```
 7 ..0000..........
 8 .00..0..........
 9 .00..0..........
10 .000............
11 .0000...........
12 ..0000..........
13 00.000..........
14 00..00..........
15 00000...........
```

Stored Digitized Character Picture

```
 0 ................
 1 ................
 2 ................
 3 ................
 4 ................
 5 ................
 6 ................
 7 ..0000..........
 8 .00..0..........
 9 .00..0..........
10 .000............
11 .0000...........
12 ..0000..........
13 00.000..........
14 00..00..........
15 00000...........
16 ................
17 ................
18 ................
19 ................
163> l
```

Expanded Character Picture

Figure 3

SAIL FONT FILE FORMAT

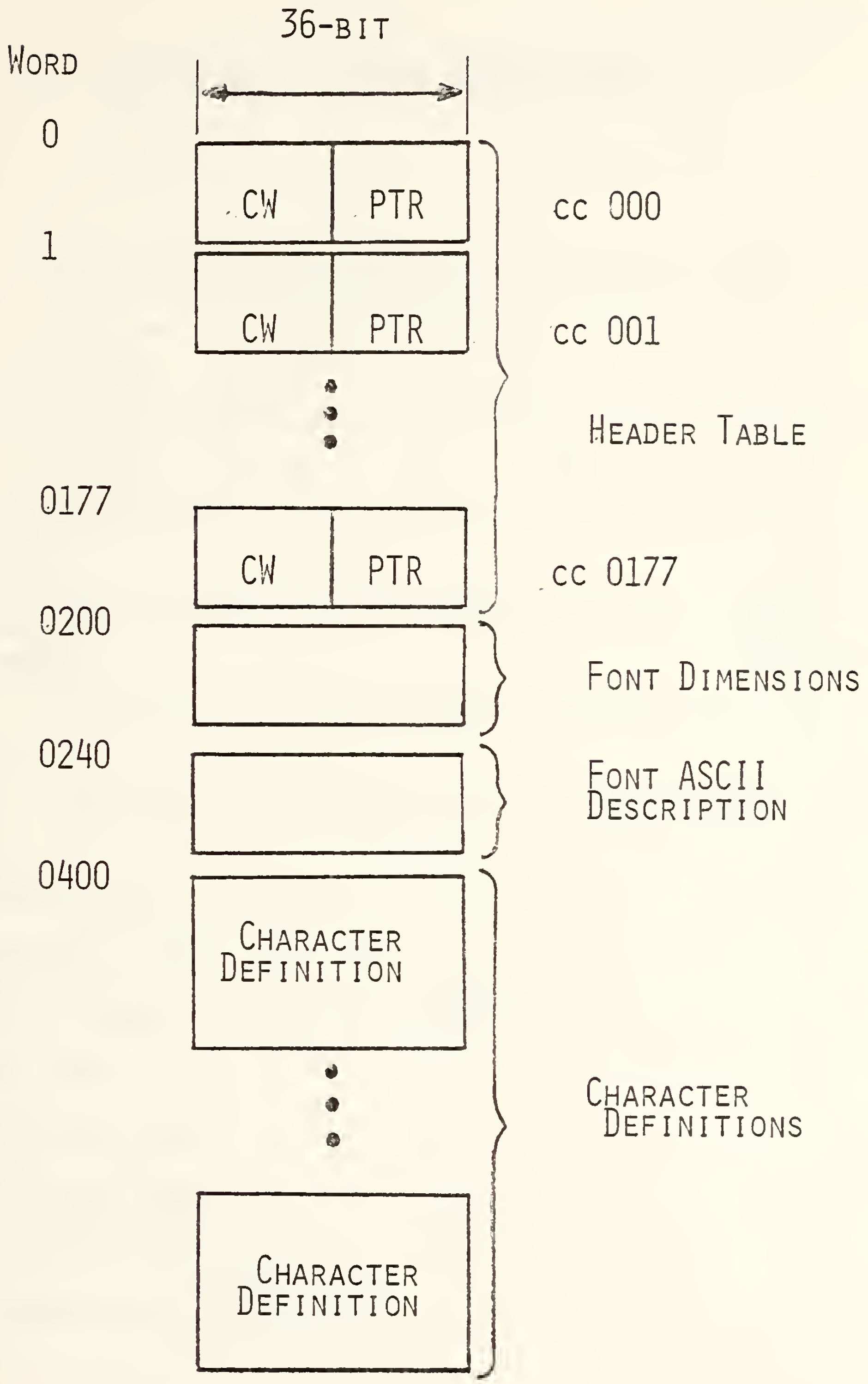
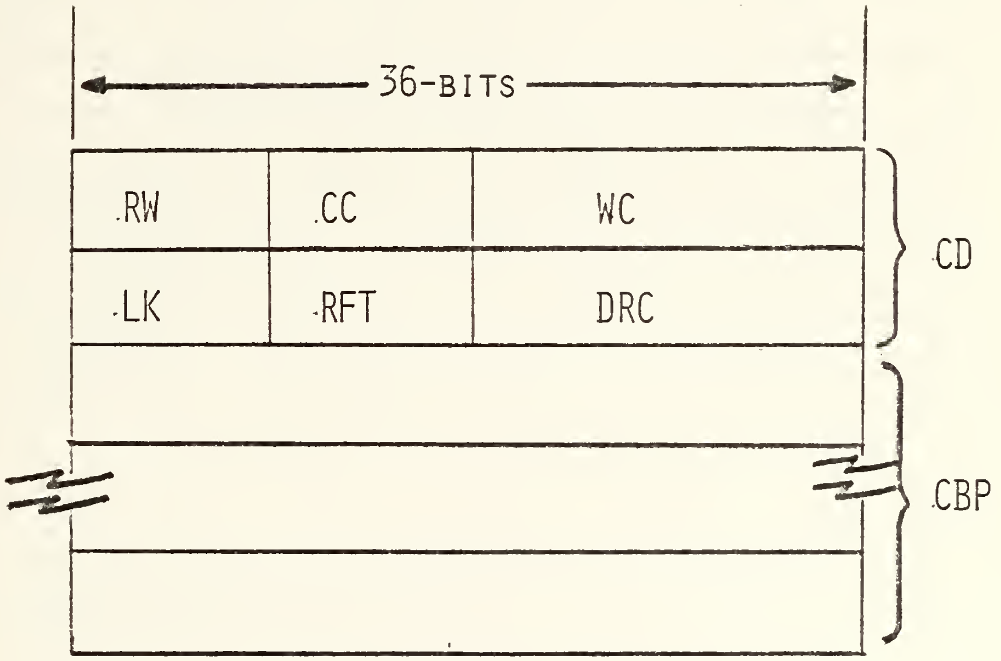


Figure 4

SAIL CHARACTER DEFINITION



RW	=	RASTER WIDTH
CC	=	CHARACTER CODE
WC	=	TOTAL NUMBER OF WORDS IN CHARACTER DEFINITION
LK	=	LEFT KERN
RFT	=	ROWS-FROM-TOP
DRC	=	DATA-ROW-COUNT
CD	=	CHARACTER DIMENSIONS
CBP	=	CHARACTER BIT PICTURE

Figure 5

4. Character Set

The ASCII and SAIL character sets are two different mappings of characters into a code (0-128). Figure 6 illustrates the differences, while figure 7 displays the complete sets. Where the ASCII set contains control characters, the SAIL set contains some additional printable characters. This situation was annoying since there were no hard-wired keyboards at NPS with which to select these characters. Consequently, to select characters occupying ASCII code positions which are not printable, text processing and typesetting programs have software escape mechanisms to get at these characters. The escape mechanism is described in Chapter 4.

SAIL/ASCII DIFFERENCES

Symbol	SAIL	ASCII
λ	'10	
_	'30	'137
~	'32	'176
≠	'33	
↑	'136	
ESC	'175	'33
}	'176	'175
BS	'177	'10
ϵ		'136
DEL		'177

Figure 6

SAIL CHARACTER SET

	0	1	2	3	4	5	6	7
000	NUL	↓	α	β	∧	¬	ε	π
010	λ	HT	LF	VT	FF	CR	∞	∂
020	⊂	⊃	∩	∪	∀	∃	⊗	↔
030	_	→	~	≠	≤	≥	≡	∨
040	SP	!	"	#	$	%	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
130	X	Y	Z	[	\	]	↑	←
140	'	a	b	c	d	e	f	g
150	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w
170	x	y	z	{	|	ESC	}	BS

ASCII CHARACTER SET

	0	1	2	3	4	5	6	7
000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL
010	BS	HT	NL	VT	NP	CR	SO	SI
020	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB
030	CAN	EM	SUB	ESC	FS	GS	RS	US
040	SP	!	"	#	$	$	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
120	X	Y	Z	[	\	]	↑	-
130	'	a	b	c	d	e	f	g
140	h	i	j	k	l	m	n	o
150	p	q	r	s	t	u	v	w
160	x	y	z	{	|	}	}	DEL

Figure 7

5. Listfont and Error Detection

As previously described, the font files read in from tape required conversion to a format more suitable to UNIX and the PDP-11. Prior to converting the files, Listfont was written. Listfont was designed to examine a Stanford font file, ignoring wasted bits and interpreting 18-bit PDP-10 halfwords as 16-bit PDP-11 full words. Listfont reads in the header table; it extracts the font dimensions and description, displaying them on the CRT screen, and proceeds to process each character definition. In processing each character definition, Listfont performs computations to ensure that, if the character and raster widths differ, there is valid kerning. Also, Listfont checks and flags nonzero left kerns of characters whose raster and character widths are equal. Additionally, using the picture accessing information, Listfont verifies picture storage. An optional "-l" argument to Listfont causes the individual character dimensions and picture to be displayed on the CRT screen. Another type of error which Listfont detects is the presence of extraneous bytes in the file.

In processing a file, Listfont counts each byte. A comparison of this byte tally with the file size indicated by an "ls -l filename" proves the absence or presence of such extraneous bytes. The time invested in the design and writing of Listfont was returned by its success in detecting errors of the above types. Two files had characters which had nonzero left kerns and identical character and raster

widths. Furthermore, two other files were found to contain several occurrences of extraneous, unused bytes. Such error detection was important in that it greatly assisted in the design of Transfile, the program to convert font files from the Stanford to the NPS format. Transfile uses the same error detection techniques and accomplishes error correction concurrently.

B. FILE CONVERSION

1. NPS Font File Format

Given the existing Stanford font file format, the design of an NPS format was not difficult. There were several criteria for the design. First, the design had to be compatible with UNIX [Ref. 11] and the PDP-11 (16-bit word processing). Second, file size needed to be minimized to facilitate typesetting in the minicomputer environment. And, third, the format needed to provide, as did Stanford's, the random accessing of character definitions. The NPS font file format is illustrated in figures 8 and 9. It is broken into three sections:

NPS FONT FILE FORMAT

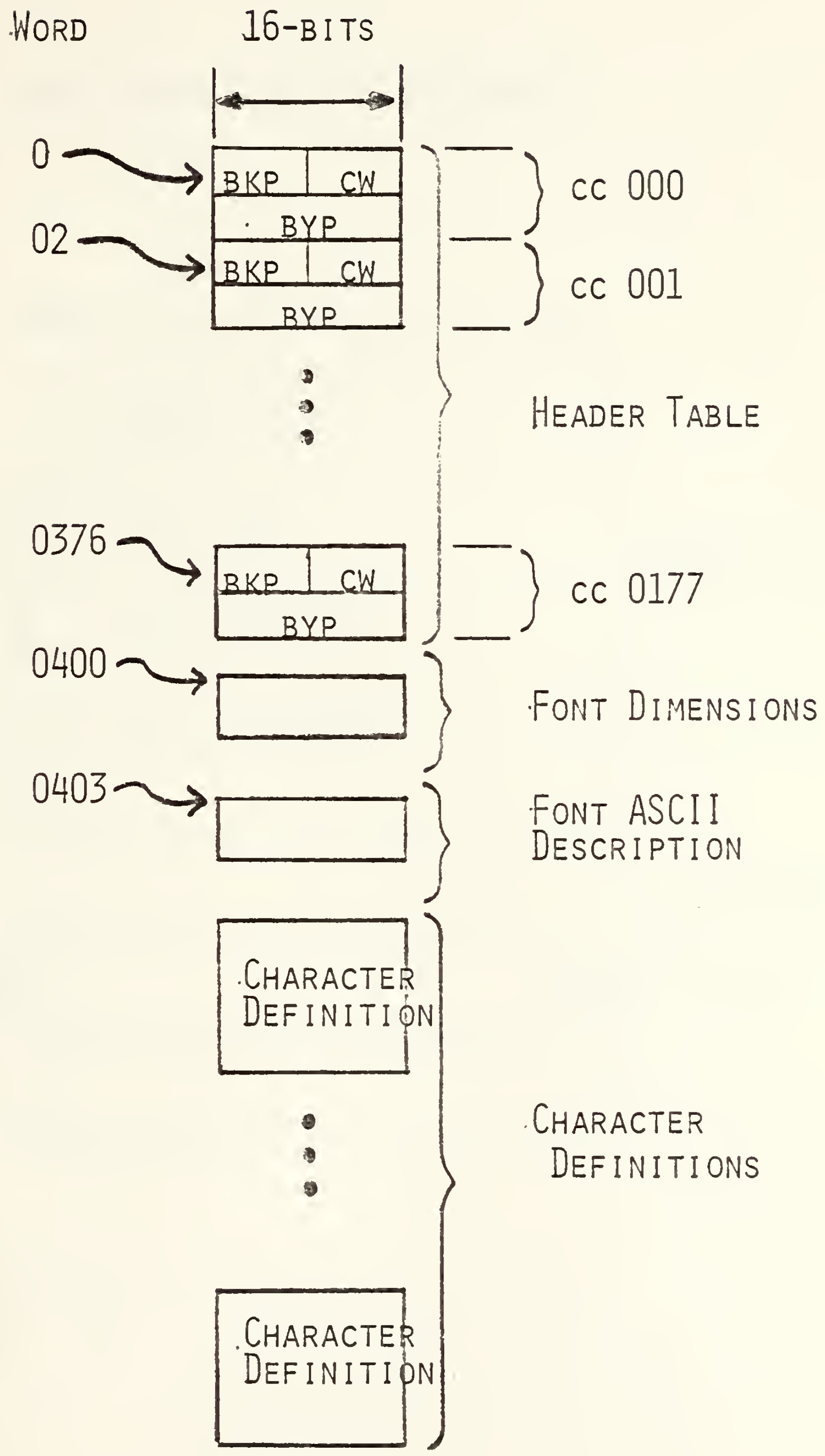


Figure 8

NPS CHARACTER DEFINITION

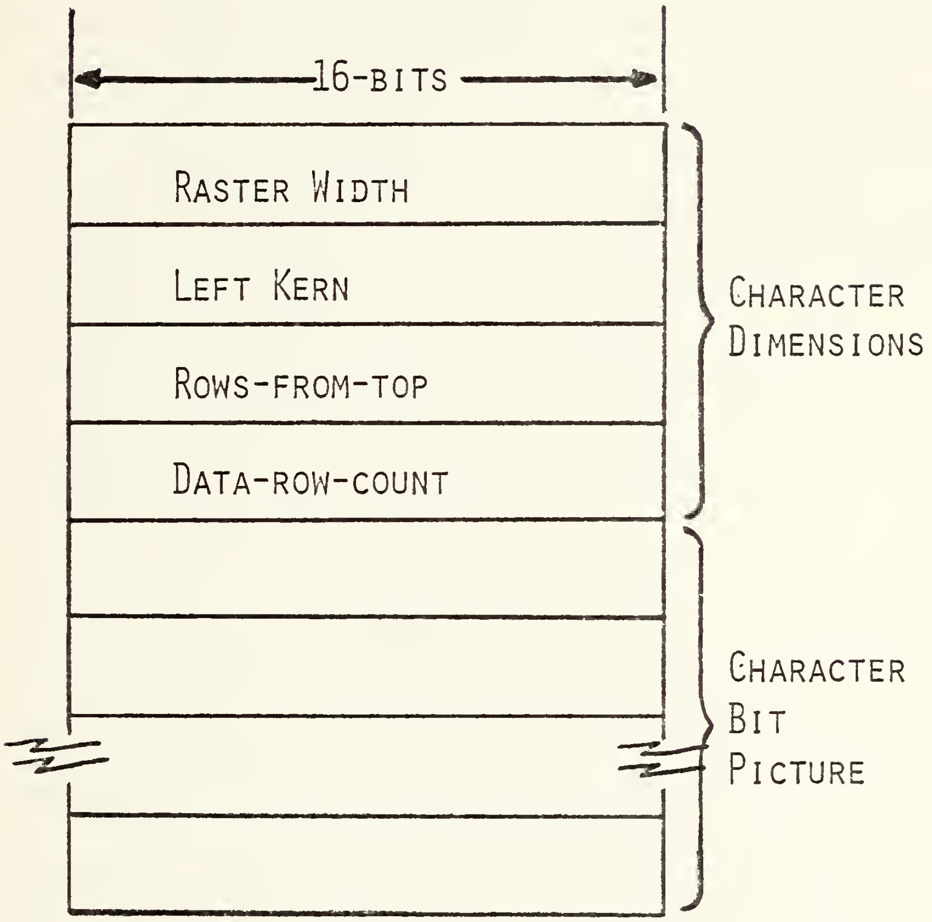


Figure 9

a. Header Table

The header table, at the beginning of the file, contains two, 16-bit words for each of the possible 128 characters in the font. The indexing mechanism to the table is the character code. The first word of each pair contains the character width in the rightmost byte. The pointer structure, indicating the location of the character definition in the file, consists of a block count (512 bytes/block) in the leftmost byte of the first word and an additional byte offset contained in the second word. A maximum block count of 255 and a maximum byte count of 32K allow for font files to approach 160K bytes. A zero in the first word in any character position in the header table implies that the particular character is not defined in the font.

b. Font Dimensions/Description

Only three dimensions are stored. Font height, maximum character width, and font logical height are each stored in words. The ASCII description follows and is stored one character per byte. It is terminated with a '\0'.

c. Character Definitions

Here, the NPS format provides substantial savings in memory. Four character dimensions are stored, each in a full word: the raster width, left kern, rft, and drc.

The raster lines which comprise the visible portion of the character picture are stored sequentially on byte boundaries following the drc.

d. File Advantages

The header table in the NPS format is twice the size of that in the Stanford version; however, each NPS character definition stores only four dimensions as opposed to the six in a Stanford character definition. This trade-off results in no real savings in memory. Significant savings occur in the storage of the raster lines of character pictures. In the Stanford version, raster lines wider than 18 pixels occupy one full 36-bit word with the following raster line beginning at the next word boundary; hence, up to 17 bits could be wasted. In the NPS version, raster lines begin on byte boundaries; therefore, no more than 7 bits will ever be wasted for any raster line. As an example, assume a fixed width font of 19 pixels is created and that 2,016 total raster lines are needed to represent all character pictures. The NPS format would require 6,048 bytes or 48,384 bits (10,080 of which would be wasted). The Stanford format would require 2,016 words or 72,576 bits (34,272 of which would be wasted). The NPS format would have stored the equivalent information in 2/3 of the memory required by the Stanford format. Berg [Ref. 2] has stated that "as a rule of thumb, for 100 printing characters at 10 point size, approximately 8,000 (16-bit) words of storage are required." SAIL10 (120 printable characters), BDR10 (120), and NONS

(96) have file sizes of 8010, 6198, and 3872 bytes respectively. The comparison is a general one in that Berg's rule specified no particular plotting density. For example, if the rule were applicable to a plotting device with a plotting density of 400 pixels per inch (twice that of the Versatec), then one could conclude that the NPS font file format generally required memory in accordance with the rule.

2. Transfile and Error Correction

Transfile was designed and written to transform font files from the Stanford format to the NPS format and, in doing so, to detect and correct any errors. Transfile takes pairs of arguments, transforming the first argument of a pair, which must name a Stanford file, to an NPS file which is given the name of the second argument of the pair. An odd number of arguments causes Transfile to exit. In transforming a file, the program first creates the NPS file and writes out a blank header table. It then examines the header table of the Stanford file to determine the number of characters in the font, reads in the font dimensions and description, and processes the character definitions. As does Listfont, Transfile ignores the two wasted high order bits of each byte and compacts 18-bit PDP-10 halfwords into 16-bit PDP-11 full words. It writes out the font dimensions and the description, if any. Transfile also writes out the NPS filename in the event that the Stanford file had no description.

In processing each character, Transfile checks dimensions to ensure compatibility and makes corrections if necessary. For example, if a character has equal character and raster widths and a nonzero left kern, then the left kern is set to zero, and the proper dimensions are written out. Listfont, as previously mentioned, detected this type of error twice. Each occurrence was in a font which had no other kerned characters; therefore, the error was corrected by ignoring the nonzero left kern, i.e., by setting it to zero. Transfile also detects unused (empty) bytes in the file, essentially throwing them away. The program keeps a running count of bytes written out and marks, in a program data structure, the starting byte address for each character definition.

After processing the last character, Transfile seeks to the beginning of the file and writes in the new header table. Upon finishing each pair of arguments, the program displays the file size in bytes. A comparison with the size indicated by an "ls -l NPSfilename" verifies a successful file transformation. Files, once transformed, decrease to between 47-83 percent of their original size. Execution times were not measured for either Listfont or Transfile as both programs were intended to be run only once on any one file.

III. EDF

A. EDF...THEN

Originally, Edf was designed by Professor Barksdale to provide the capability of creating and editing a particular class of fixed width fonts, all characters being 16 pixels wide and 20 pixels high. Edf was an interactive program implemented in the programming language C. In its edit mode, Edf would read an entire font file into a character array (128x40). Each character definition was accessed by its character code (0-127), and its bit picture consisted of the next 40 bytes, two bytes representing one raster line in the character picture. The simple font design and data structure facilitated easy character definition accessing for listing, editing, or deleting, etc. In the create mode, the array (128x40) was cleared, and the user began with all characters having blank pictures. Edf possesed an efficient command handling module and input several and display routines. Using these routines as a skeleton, Edf was modified to edit and create fixed and variable width fonts of different sizes.

B. MODIFICATION REQUIREMENTS

Prior to modifying Edf to manipulate variable width fonts, certain requirements were first identified:

1. File Format

Edf needed to be able to interface with the newly designed font file format. It had to be able to access character definitions, font dimensions and description, and it had to be able to write out edited or created fonts in this new format.

2. Commands

From the set of commands available in the original version of Edf, a minimal subset of commands needed to be implemented. This subset could be defined by excluding the "nice to have" commands. The commands available under the improved version of Edf are described later in this chapter.

3. Memory Requirements

Edf needed to be able to deal with fluctuating memory requirements due to the dynamic sizing of characters in the fixed and variable width fonts. Static data structures could not provide such flexibility. Specifically, a buffer, large enough to hold the biggest character definition, would be needed. Additionally, Edf would have to be able to store modified character definitions of varying sizes until the edited or created file could be written out.

4. Edit Status

The editing and creating of variable and fixed width fonts increases the length of the interactive session, and the added complexity of varying character dimensions can often cause a user to forget what has been accomplished and what remains to be done during the edit session. Edf needed to be able to provide some table or display, showing the status of each character in the font, i.e., undefined, defined but unmodified, modified, deleted, etc.

5. Dimensions

In addition to being able to change character pictures, the user must be able to change font and character dimensions, and any change must be checked to ensure that it is a valid, reasonable one. As examples, a user must not be allowed to increase the character width of a particular character to a value greater than its raster width, nor must he be allowed to change a font's height to a negative value. Edf must be able to compute the new rft and drc of a modified character picture; however, Edf should not be responsible for ensuring that the modified picture is accurately described by all character dimensions. For instance, if a user were to change the picture of the character "a", making it shorter and skinnier, Edf must be able to compute and update the rft and drc. The user would then be responsible for making the appropriate adjustments to the character and raster widths of "a". Such restrictions are necessary to

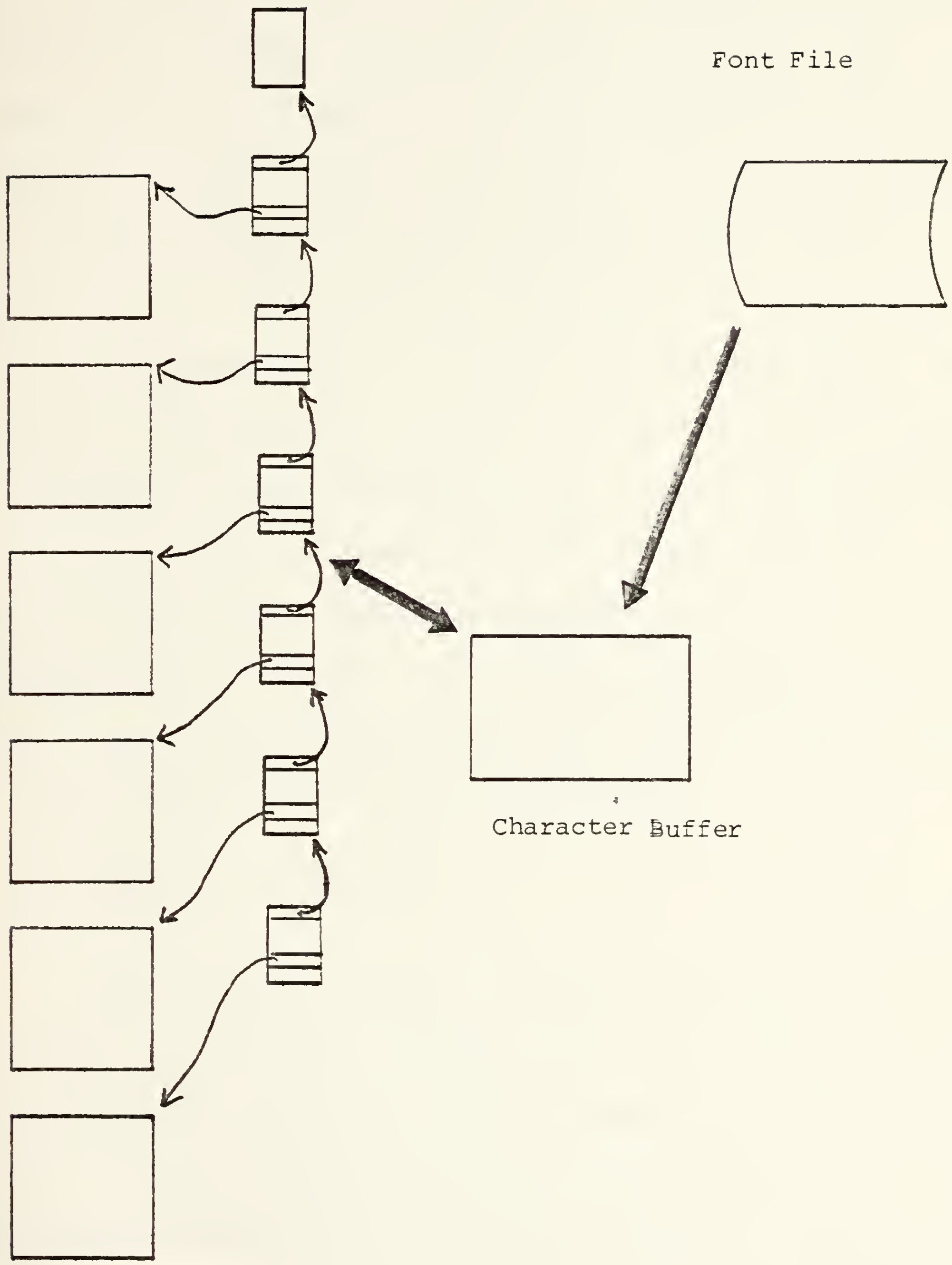
limit program overhead.

C. CONCEPTS AND TECHNIQUES

1. Concepts

There are several concepts which make up the configuration of the interactive font edit/creation process. A character buffer holds the character definition being modified, a linked list manages the modified character definitions, and a file, if in the edit mode, represents the information (character definitions) requiring changes. The UNIX system routine Alloc(II) [Ref. 13] provides temporary memory to store modified charcter definitions. Figure 10 illustrates the file/work area configuration.

Edf FILE/WORK AREA



Modified Character Definitions

Figure 10

2. Techniques

a. Current Character

"Current Character" (cc) is a pointer to a character position (0-127) in the font being edited or created. Any command takes the character definition pointed to by cc as its operand. The character definition referred to by cc is never loaded into the character buffer unless some command requires it. Edf prompts with the octal value of cc, initially 0, followed by ">". The current character may be incremented, decremented, or set to any value in the range 0-127. Wraparound occurs automatically when incrementing above 127 or when decrementing below 0. Whenever cc changes, Edf determines, before executing any command, if the character definition in the buffer has been modified. If so, Edf reads the modified definition out to the linked list and then executes any awaiting command.

b. Character Buffer

The character buffer is 4000 bytes long and is large enough to hold the biggest character allowed within the limitations of font height and character width. Edf will edit or create fonts up to 120 pixels (about 42 point) in height and characters up to 255 pixels wide. There is a routine responsible for loading character definitions into the character buffer. Whenever a command requires a definition, this routine will first inspect a global flag which indicates if the definition pointed to by cc is already in

the buffer. If the definition pointed to by cc, the operand for any command, is not in the buffer, this routine will load it into the buffer from one of two places. First, definitions which have been previously modified or definitions in a font being created will be loaded into the buffer from the linked list. Otherwise; the definition is accessed and loaded from the file being edited. In loading the buffer, the character dimensions (raster width, left kern, rft, and drc) are stored in the first eight bytes. Then, using the rft and the raster width, the required number of blank lines are inserted into the buffer. For example, a raster width of 17 requires 3 bytes for storage. If the rft were 4, then 4 blank lines or 12(4x3) zero bytes would be inserted. Next, the routine uses the drc and raster width to read the digitized portion of the character picture into the buffer, and, finally, using the rft, drc, and font height, it computes and inserts the necessary number of blank lines needed to complete the character picture.

c. Character Picture

The character picture was expanded when the character definition was read into the buffer. The picture is accessed beginning at the ninth byte and is displayed on the CRT screen with line numbers from 0 to "font height-1". The width of the matrix in which the character picture is displayed is equal to the number of bits in the bytes required to store a raster line; therefore, unless a character's raster width is a multiple of 8, its displayed

picture will make the character appear wider than normal, i.e., if a character's raster width is 17 and the font height is 20, then the character picture will be displayed in a 20x24 matrix, since 3 bytes (24 bits) are required to hold a raster line.

d. Linked List

The linked list contains a node for each modified charcter definition. Each node contains the character code, which is the ordering criteria for the list (the lowest code is placed at the head of the list), a pointer to the block of memory (provided by Alloc(II)) holding the character definition, the status of the character's modification ("m"-modified, "i"-included, or "d"-deleted), and a pointer to the next node in the list. A dummy node with a character code of 32677 marks the end of the list.

e. Font/Character Dimensions

Having added or changed a character picture, the user may want to change or may need to change character dimensions. Also, he may wish to change font dimensions or the font description. There is an interactive module which is quite versatile in allowing these changes. The module is described in the command descriptions. It displays a set of instructions upon entry and has a unique prompting symbol.

f. Writing Out a Font File

In writing out a file, Edf first writes a blank header table followed by the font dimensions and description. Then, beginning at character code 0, Edf incorporates modified character definitions from the linked list with unchanged definitions from the file. It maintains a byte count, in 512-byte blocks and bytes, of bytes written. Once the last definition has been written out, Edf seeks to the beginning of the new file and writes in the new header table. Edf will remove the new file from the directory if no character definitions were written, i.e., the user wrote out a font in which he had deleted all characters during the edit session. As a final gesture, Edf displays the new file byte size in decimal before quitting.

D. CAPABILITIES

1. Invoking Edf

The current version of Edf is considerably larger than its predecessor, a growth resulting from the addition of modules to manipulate the more complex and more dynamic format of the new font files. Creating a font may be accomplished by one of several means. First, a call to Edf with no arguments indicates that the user desires to create a font from scratch. The user must specify the characteristics of the new font and then use the "a" (add) command to create specific characters at each character position. Re-

peating this process for 128 characters can become exceedingly tedious. A more efficient option is to create only a few new characters and to then use the "i" (include) command to include other characters from a compatible font. A third option, somewhat similar to the second, is to use the "d" (delete) command to remove unwanted characters from a selected base font.

To edit an existing digitized font file, Edf requires an argument consisting of either a font file name or a complete path name. In the first case, the font editor assumes that the font is located on the directory "/.fonts.01/font/" and prepends that string to the argument before issuing a system call to open that file. If a complete path name is used, Edf will open that font file. If the font file is missing or if the font file contains invalid information, then Edf will exit with an appropriate error message. A Hershey font [Ref. 5], digitized to any desired size and subject to the limitations discussed later, can also be edited. References 5 and 7 provide excellent descriptions of the Hershey fonts. Some examples of valid calls to Edf are listed below:

```
                              edf
```

This indicates that the user desires to create his own font. He may give it any name when he writes it out, ending the edit session.

```
edf  SIGN41
```

The user wants to edit SIGN41 on "/.fonts.01/font".

```
edf  /usr/doyle/fonts/HTR42
```

The user wants to edit an existing Hershey font file called HTR42, a Triplex Roman font at 42 point, on directory "/usr/doyle/fonts/".

```
edf  HSR20
```

The user wants to edit an existing Hershey font file called HSR20, a Simplex Roman font at 20 point, on directory "/.fonts.01/font/".

```
edf  -HGE 36
```

The user wants to create a Hershey font file in the Gothic English type at 36 point. He may write it to any directory after it has been digitized.

```
edf  -HCS
```

The user wants to create a Hershey font file in Complex Script type. The point size defaults to 10 point, and the font may be written to any directory at the conclusion of the edit session.

2. Commands

The basic command line consists of three parts: the current character selector, the command itself, and arguments, if any, to the command.

a) <number>

Change the current character to <number>. The number may be octal (preceded by a zero) or decimal. Any number greater than 127 is converted to 0, and anything less than 0 is converted to 127. Any command may appended to <number>. The effect is to change the current character first and then to execute the appended command.

Examples: 0176, 0, 161, 78c 0 25, 16a.

b) +¦-

Increment (decrement) the current character. Wrap-around occurs as in <number> above. Either <+> or <-> may be used but not both on the same command line. Any command may be appended to either, and the effect is to increment (decrement) the current character first and then execute the command. Only one "+" or "-" may be used on a command line.

Examples: +1, -, +, +e, +c0 40.

c) [<number>]¦[+]¦[-]a

Add a new character to the font at the current character position. The "a"(add) command is complex. A "p"(parameter) command is executed automatically. The displayed instructions to input the dimensions of the new character must be followed. The new character is being defined at the current character. After exiting the parameter command loop, the user may use the "c"(change), "e"(edit), "s"(shift), or "l"(list) commands to form the desired character picture. The character buffer has previously been zeroed. If the user uses <number>, "+", or "-" to change the current character before he is satisfied with the new character picture, the unsatisfactory picture is stored. If this happens, the character picture may be relisted and changed.

Examples: +a, -a, 056a, 19a, a.

d) [<number>]¦[+]¦[-]c[<number>] [<number>]

Change lines "s" through "e", prompting for each line. "c" alone sets "s" to 0 and "e" to "height-1". "c" followed by one number sets both "s" and "e" to that number. "c" with two numbers sets "s" and "e" accordingly. The numbers may be octal or decimal, and a space is required between two numbers.

Examples: +c, -c0 10, 077c 1 044, c, +c 10.

e) d[<number>] [<number>] font file

Delete characters "s" through "e". "d" alone sets "s" to 0 and "e" to 127, effectively deleting the entire font. "d" with a single number deletes that character code. "d" with two numbers deletes "s" through "e" inclusive. Numbers may be octal or decimal, and a space is required between two numbers.

Examples: d, d5, d 0176, d 0 057.

f) [<number>]¦[+]¦[-]e[<number>] [<number>]

Edit lines "s" through "e", prompting for each line. "s" and "e" are set as in "c"(change). While editing a line, "cntl-d" completes the line as it was. This command uses the NPS line-editor functions in the terminal handler.

Examples: e, 077e0 10, +e 3 5, -e, 017e 12.

g) f

Turn on (off) a flag controlling the display of character dimensions. Once turned on, character dimensions are displayed every time a character definition is fetched. Displaying is turned off by a subsequent "f". "f" may be prepended to any command.

Examples: f, fl, +fe 0 10, 0176fl 0 10.

h) i[<number>] [<number>] filename

Include characters "s" through "e" from the font file "filename". "s" and "e" are set as in the "d"(delete) command. If the font file being edited or created and "filename" are not compatible, then the include will not occur. Subsequent uses of "i" do not require "filename"; unless, of course, the user wishes to include from another font file.

Examples: i 0 057 BDJ8, i HCS20, i.

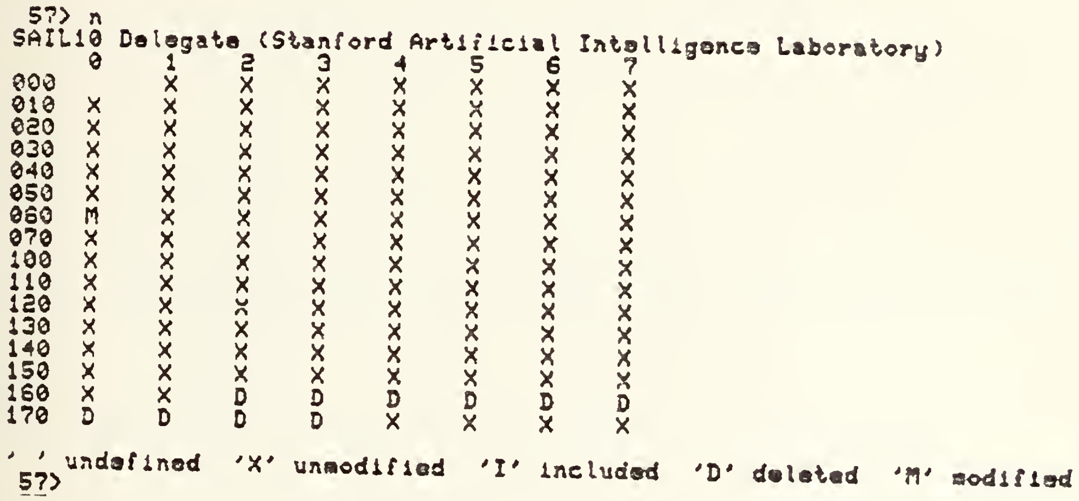
i) [<number>]¦[+]¦[-]l [<number>] [<number>]

List lines "s" through "e" of the current character. "s" and "e" are set as in "c"(change).

Examples: +l 0 10, -l, l, 076l, l 12.

j) n

Display the font description and a table reflecting the status of the edit session. The table provides an excellent means of managing edit work. Figure 11 illustrates the results of executing an "n" command during an edit session.



```
 57> n
SAIL10 Delegate (Stanford Artificial Intelligence Laboratory)
       0     1     2     3     4     5     6     7
000          X     X     X     X     X     X     X
010    X     X     X     X     X     X     X     X
020    X     X     X     X     X     X     X     X
030    X     X     X     X     X     X     X     X
040    X     X     X     X     X     X     X     X
050    X     X     X     X     X     X     X     X
060    M     X     X     X     X     X     X     X
070    X     X     X     X     X     X     X     X
100    X     X     X     X     X     X     X     X
110    X     X     X     X     X     X     X     X
120    X     X     X     X     X     X     X     X
130    X     X     X     X     X     X     X     X
140    X     X     X     X     X     X     X     X
150    X     X     X     X     X     X     X     X
160    X     X     D     D     D     D     D     D
170    D     D     D     D     X     X     X     X

' ' undefined  'X' unmodified  'I' included  'D' deleted  'M' modified
 57>
```

Figure 11

Example: n.

k) p

The "p"(parameter) command executes an interactive module of Edf which allows the modification of character and

font dimensions and description. A set of instructions will be displayed and may be recalled if required. This module is quite versatile. The user must keep in mind that character and font dimensions are being changed, not character pictures.

Example: p

l) q

Quit warns once if changes have been made and not written out; otherwise, it exits, closing any open files.

Example: q.

m) [<number>]¦[+]¦[-]s l¦r¦u¦d [<number>] [<number>]

Shift lines "s" through "e" one pixel left(l), right(r), up(u), or down(d). The resulting lines are automatically displayed. "s" and "e" are set as in "c"(change).

Examples: +s10 10, 044su 10, sr, -sd.

n) w filename

Write out the font file being edited or created to "filename". "w" must have a "filename" and will not allow the user to write to the font file being edited. "w" displays the byte size, in decimal, of "filename" and then performs a "q"(quit). Writing out a font file takes longer than writing out a normal file.

Examples: w temp, w /.fonts.01/font/HCI20.

o) <rubout>¦<break>

Either key causes an interrupt which is trapped. Whatever command was executing is stopped, the previous environment restored (the command loop is reentered), and the user may continue. Neither key undoes anything; they merely give a mechanism for killing commands without killing the program.

E. LIMITATIONS

There are two types of limitations to Edf. First, there are "nice to have" type commands such as folding character pictures, italicizing fonts, and producing bold fonts which were not included due to time constraints but which could easily be added in the future. Second, Edf has not had a thorough testing. There are many checks throughout the program which were included to detect bad font files and to

prevent the program from abnormal termination. Edf is good at screening commands and at flagging bad ones. Although possible to string some commands together on one command line, some combinations are bound to produce strange results. The user should combine commands only as described in the preceding section. Despite its limitations, Edf is an extremely useful tool.

IV. TYPESETTING TOOLS

The user's manual [Ref. 7] provides detailed instructions on the use of the two typesetting tools described in this chapter.

A. PRFONT

1. Design

Prfont was designed as a final test of a digitized font. If a font, when displayed by Prfont, appears ragged, then it is not yet satisfactory for use in typesetting. Prfont displays an entire font, setting characters on horizontal lines in their collating sequence. To do this, Prfont reads in the header table and font dimensions from the font file, checking for invalid font dimensions. First, using the Versatec simultaneous printer/plotter mode, the font name is centered above where the font will be displayed. Prfont then runs through the header table acquiring enough characters for a row. Once a row has been filled, Prfont fills plot buffers with the digitized pictures of the collected characters. Plot buffers, once filled, are sent to the Versatec one at a time. Once a number of plot buffers equal to the height of the font have been sent, the line of character pictures is complete, and one line of characters has been set. Prfont then plots 5

blank plot lines to provide character line spacing. Before continuing, the program frees the allocated memory (from Alloc(II)) that it acquired to hold the character definitions awaiting plotting. Prfont frees this memory in the reverse order in which it was requested. This reverse order of freeing is important. During the testing of Prfont, certain sequences of memory allocations, if not freed in reverse order, caused an abnormal program termination when the program was later requesting additional memory (in the system routine Alloc(II)). This problem became much more complex in Signmkr where certain characters were used several times on a line. Prfont then gets another row of characters, continuing the process until all characters in the font have been displayed. In setting character pictures, Prfont sets all the bytes used to store the bit picture. For example, if a character has a raster width of 17, then 3 bytes(24 bits) are set in the plot buffer, as opposed to the setting of the first 17 bits alone. Setting pictures by bytes as opposed to bits greatly speeds the process of filling plot buffers while producing the same character pictures.

2. Features

Prfont takes multiple arguments, either font names or full pathnames. Prfont ensures a one and one half inch margin at the top of the page and one inch margins elsewhere. Furthermore, Prfont looks ahead to ensure that the next font to be displayed will fit on the current page,

causing a page eject if sufficient room does not exist. Prfont also takes an optional numeric argument. This argument must be the the first argument and must be preceded by a "-". The argument, any number from 1 to 264, resets the width of the display field in plot bytes. Often, in an extremely large font or on days when UNIX is servicing many users, Alloc(II) will be unable to provide the memory required to hold the character definitions awaiting plotting. If this situation occurs, the program exits, displaying a message to rerun with a narrower display field. Such a field would hold fewer characters and, therefore, require less memory. The default pagewidth, or display area, is 216 plot bytes.

B. SIGNMKR

1. Design

As the thesis objectives requiring the modifications of Troff and Vts were not attained, this author desired some means, however limited, of setting text with the adapted fonts. With that objective in mind, Signmkr was designed. Signmkr reads lines from a text file interspersed with a limited set of text processing commands and sets the text, according to the commands, in the selected fonts. Briefly, the design includes both text processing and typesetting functions. The program is a novelty; it is more suited to making signs than for producing documents.

Signmkr loads a default font, SAIL10, and commences to read characters into a text buffer until a '\n' is encountered. Further described in the command listings in the following section, there is an escape mechanism to provide breakpoints at which certain text processing and typesetting tasks are performed, e.g., loading a new font, centering a line of text, specifying a character code for a printable SAIL character in an ASCII control character position, etc. Characters are transferred one by one from the text buffer to a print buffer. During this transfer a plot width is maintained and escape options handled. Once a '\n' has been found in the text buffer or the plot width of the print buffer exceeds the pagewidth (the concept of pagewidth is the same as that in Prfont), transferring stops, and characters in the print buffer are expanded into multiple plot buffers by the insertion of their digitized pictures. Again, the plot buffers are sent to the Versatec one by one; however, in Signmkr, digitized pictures are set bit by bit as opposed to Prfont's byte by byte picture setting, and pagewidth is measured in bits as opposed to bytes. The overhead involved in extracting the bits from the bytes storing a raster line, has been minimized. Only one procedure call is required to obtain the next bit in a raster line.

If a user has placed a character in a file and if the characters are being set in a font in which that particular character is undefined, Signmkr will automatically in-

sert the blank (040) character in its place. If the font has no blank character, Signmkr exits. Whenever Signmkr cannot handle requests, it displays diagnostics and the line being processed before exiting.

Example: this source produces this.

```
cαααααπααααααɔ          cαααααπααααααɔ
~  #   ~  #  αβαααααα→~  B   ~ NIL ~
%ααβαα∀αααααα$          %ααααα∀αααααα$
   ~   cαααααπααααααɔ
   %α→~   A   ~ NIL ~
      %ααααα∀αααααα$
```

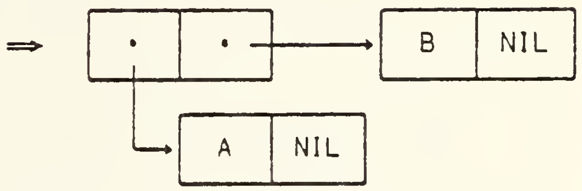


Figure 12

2. Features

Figures 12 and 13 are examples of Signmkr's capabilities, and, in fact, figure 13 is an excellent description of Signmkr in itself. Some of the figures in this thesis and most of the figure titles were set by Signmkr. The various commands to Signmkr are summarized below. "ESC" is the ASCII escape character (033).

a) ESCc < one line of text >

The "center" command centers one and only one line of text, and that line is the line immediately following the

command. The user must use this command for each line to be centered. If a line is too long to be centered, then Signmkr will inform the user of this fact and ignore the line.

b) ESCf<fontname>

This command allows the user to change the font being used for typesetting; it must be used only at the head of a line or on a line by itself. Full pathnames are acceptable. A blank must not be left between the command and the new font name.

c) ESCpg\n

This is the "pagebreak" command and is similar to the ".bp" command used in NROFF. It sends a form-feed signal to the Versatec. The command should be used on a line by itself.

d) ESCpp\n

The "begin paragraph" command indents the text line for paragraphing. The size of the indent is determined by the size of the current font. Like the "pagebreak" command, it should be on a line by itself.

e) ESCs<number>

The "space" command inserts blank lines within the text. The height of the blank line is equal to the font height. A blank must not be left between the command and the number. The number may be octal (leading 0) or decimal.

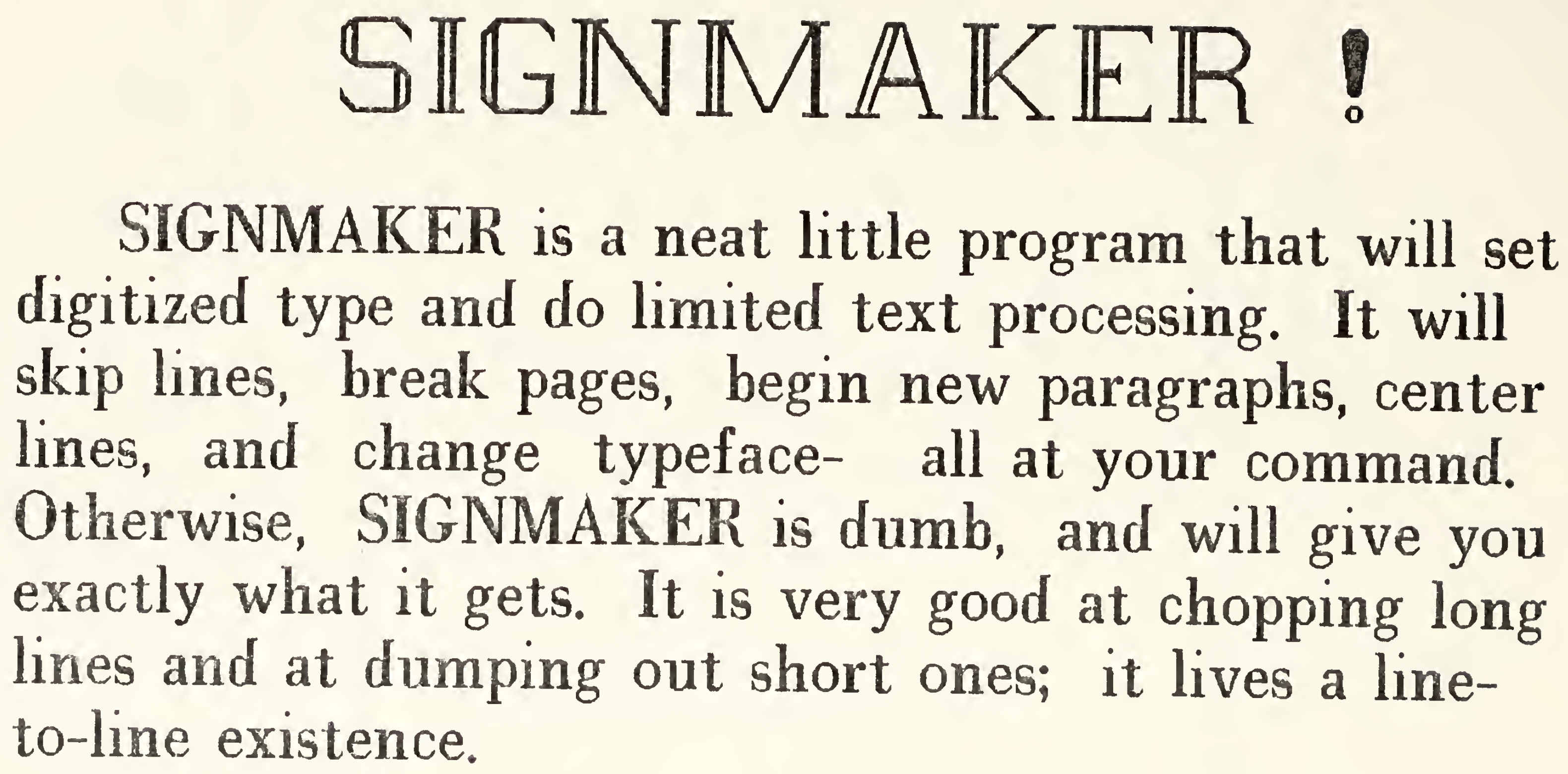
f) ESCo<number>

This command specifies a character by its character code within the current font. The command may be used at any point within a line, but it must not contain blanks. This command is useful in accessing a character from a SAIL font whose character code corresponds to a control character in ASCII. Numbers may be octal (leading 0) or decimal.

Users with previous experience with text processing programs should have no trouble in adapting to Signmkr. However, caution should be exercised when using the "ESCpp" (paragraph) and "ESCf" (change fonts) commands at the same point in the input file. The two sequences of input lines

```
(a) ESCf BDR8              (b) ESCf BDR8
    ESCpp\n                    ESCf HTR30\n
    ESCf HTR30\n               ESCpp\n
    < input text >             < input text >
```

are not identical. Sequence (a) will set up the indentation for the next paragraph assuming a font height of 8 point, but the text will actually be set in 30 point type, so the indentation will not be obvious. Sequence (b) changes the font height to 30 point and then indents based on that height.



SIGNMAKER !

SIGNMAKER is a neat little program that will set digitized type and do limited text processing. It will skip lines, break pages, begin new paragraphs, center lines, and change typeface- all at your command. Otherwise, SIGNMAKER is dumb, and will give you exactly what it gets. It is very good at chopping long lines and at dumping out short ones; it lives a line-to-line existence.

Figure 13

V. CONCLUSIONS

A. ATTAINMENT OF THESIS OBJECTIVES

In retrospect, the thesis effort may be divided into three main areas:

1. Data Base of Digitized Fonts

First, a data base of digitized fonts for a 16-bit environment was created. This accomplishment encompassed the first two thesis objectives listed in the Introduction, the design of a UNIX compatible font file format and the conversion of the thirty-four SAIL fonts to this format. This effort began in early February, 1977 and was completed in late March. Considerable time was spent in designing and programming Listfont. Listfont provided for the processing of the raw data, the Stanford font files on tape. After designing and programing Listfont, this author was thoroughly familiar with the concepts involved in storing digitized character definitions and was aware of several errors in the existing font files. This awareness was invaluable in designing a compact font file format for use under UNIX and in designing Transfile, the program to correct errors while transforming SAIL fonts to the NPS format. The resulting files represent a variety of different software type for use in computer typesetting.

2. Software Tool Development

The second area of the thesis effort consisted of completing thesis objectives three through six: the redesign of Edf to edit and create fixed and variable width fonts, the design of Prfont to display fonts, and the design of Signmkr to set text in the digitized fonts. None of the many problems encountered in program design required the modification of the font file format initially designed. The file design was such that character pictures were easily accessible, and programs could often use routines from previously designed programs with only minor tailoring.

3. Documentation

The third and final phase of the thesis effort was the documentation. First, a user's manual was written (co-authored) [Refs. 5 and 7]. The manual was designed for a student with moderate experience with UNIX, no experience in computer typesetting, and a desire to pursue further development of computer typesetting under UNIX. Second, the thesis documents the total effort, focusing mainly on program design. During this final phase, the author came to several conclusions concerning computer typesetting under UNIX and computer typesetting in general. In the former case, there is great potential for experimentation in the design of a software oriented computer typesetting environment, a software environment which could conceivably be modified to function on different computer systems using

different printing devices. In the latter case, there is great potential in printing-related industries for increased profits and lower machine maintenance costs.

B. COMPUTER TYPESETTING UNDER UNIX

Although all of the programs could be improved, as is discussed later, the system software is efficient, and the algorithms could be reprogrammed to adapt the system to another computer or, under UNIX, to drive a higher speed plotting device. To gain some appreciation for the time required to set type under the present system, "THE QUICK, BROWN FOX JUMPED OVER THE LAZY WHITE DOG." was set in increasing font sizes. The timed results are displayed in Table 3.

FONT	REAL	SYSTEM	INPUT/OUTPUT
BDJ8	18.0	0.5	2.8
BDR10	28.0	0.5	3.0
SAIL10	27.0	0.8	2.7
BDR15	27.0	1.5	3.6
SIGN22	35.0	3.2	3.8
BDR25	34.0	2.6	4.6
SIGN41	44.0	10.6	5.6

Times are in seconds.

Table 3

By examining Table 3, two conclusions are obvious. First, system and input/output times are dependent on font height. Secondly, given the above times for the setting of one sentence, the production of large documents would be unreasonable. Slow typesetting times are caused by the low plot speed of the Versatec, and the constant demands on the PDP-11's unibus design which services all users and peripheral devices. Figure 21 of Appendix A required 32.5 seconds of system time and 26.6 seconds of input/output time. In summary, UNIX has provided an excellent environment for the design of a system of programs to effect computer typesetting; however, UNIX is by no means prepared to provide the environment needed to continuously operate such a system.

C. FUTURE MODIFICATIONS

The results of this thesis and the efforts documented in reference 5 are that UNIX now posseses a large data base of fixed and variable width fonts and three significant tools for further development of the system. Troff and Vts have not been modified, and, until they are, computer typesetting under UNIX lacks its potential capability. Considering that the pre-thesis system configuration still exists for the original four fonts, the expanded font library and improved tools represent a significant enhancement. This author recommends that futher development to enrich the system be conducted in the following areas:

1. Troff

Modify Troff to process text files to be set in any of the fonts in the present library. The major effort in this area is the design of a scheme for Troff to compute character widths from a font name and height. Troff should produce a file to be processed by the virtual typesetter, Vts.

2. Vts

Modify Vts to set fixed and variable width fonts stored in the NPS font file format.

3. Software Tools

Although useful in their present forms, additional options could be addec to Edf and Signmkr. First, the capabilities of producing italicized and bold fonts in Edf from a roman font would be a significant improvement. Second, although its place in the computer typesetting system will always remain that of a novelty, some additional text formatting options in Signmkr would make it a more useful tool. Both Prfont and Signmkr can be made to execute more rapidly by filling and sending groups of plot buffers to the Versatec as opposed to the present design of transmitting plot buffers one at a time, and, in all three programs, the number of disk reads for each character definition access could be reduced from five to two. Presently, the complete character definition is accessed by seeking to and reading the raster width; three subsequent "reads" obtain the left kern, rft, and drc, respectively. After some computations, the entire bit picture can then be read into program memory (the fifth "read"). Instead, by seeking to the definition and reading all four dimensions (8 bytes) into a buffer, the bit picture can be read into program memory, after some computations, in a second "read". Thus, the number of "reads" per character access is cut from 5 to 2.

4. Kerning

The concept of kerning should not be implemented until Troff and Vts have been fully integrated into the fixed

and variable width font environment. When implemented, consideration should be given to either modifying Edf or creating a separate program to provide the ability to display pairs of characters with the kerning effect.

5. Plot Capability

As a final enhancement, both Troff and Vts should be modified to process textual and graphical information from the same file, allowing for limited graphical displays in a primarily textual document. This modification demands efficient use of memory as the Versatec cannot reverse paper movement, and Vts must be able to store information in "looking ahead" to complete graphical displays. The need for computer typesetting systems to handle both graphical and textual data is well documented and such systems provide great versatility over others where the two types must be treated separately. For example, as early as 1963, the U.S. Government Printing Office issued a request for a typesetting system based on photocomposition. One of the requirements was an ability to handle randomly occurring graphic formats in text documents [Ref. 14].

APPENDIX A FONT DESCRIPTIONS

The SAIL fonts are displayed on the following paaes. The displays were produced by Prfont and are in the same order as the listina in Table 1 of Chapter 1. The final page of the appendix was set by Signmkr and is incluaed to illustrate the contrast amona the fonts. A comparision of the SAIL fonts displayed on the following pages and those displays in reference 12 reveal added characters in the NPS versions. The additions were made at Stanford after the publication of reference 12. The additional characters have not oeen removed.

BDJ8

↓∝ βΛ¬ ϵ πλ ∬∞∂⊂⊃∩∪∀∃⊗↔_→~≠ ≤≥≡∨ !" #$%&'()*+,-./0123456789:;<=>? @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←'abcdefghijklmnopq
rstuvwxyz{|◊}

BDR10

↓∝βΛ¬ϵπλ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←'abc
defghijklmnopqrstuvwxyz{|~}

BDI10

↓∝βΛ¬ϵπλ∫∬∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ *!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]*
↑←'*abcdefg hi jkl mnopqrstuvwxyz*{|~}

BDJ10

↓∝βΛ¬ϵπλ∫ ±∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@*ABCDEFGHIJKLMNOPQRSTUVWXYZ*[\]↑←
'*abcdefg hi jkl mnopqrstuvwxyz*{|◊}

BDR10X

↓∝βΛ¬ϵπλ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←'abc
defghijklmnopqrstuvwxyz{|~}?

BDR12

↓∝βΛ¬ϵπλ   ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWX
YZ[\]↑←'abcdefghijklmnopqrstuvwxyz{|◊}

Figure 14

BDI12

↓∝βΛ¬∈πλ ff ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRS
TUVW XYZ[\]↑←'abcdef ghijklmnopqrstuvwxyz{|~}

BDB12

↓∝βΛ¬∈πλ ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVW
XYZ[\]↑←'abcdefghijklmnopqrstuvwxyz{|◊}

BDR15

↓∝βΛ¬∈πλ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQ
RSTUVWXYZ[\]↑←'abcdefghijklmnopqrstuvwxyz{|◊}

BDI15

↓∝βΛ¬∈πλff∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDE
FGHIJKLMNOPQRSTUVW XYZ[\]↑←'abcdefghijklmnopqrstuvwxyz{|~}

BDR25

↓∝βΛ¬∈πλ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;
<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←'abcd
efghijklmnopqrstuvwxyz{|fl}

Figure 15

NONSI

```
▒_~ ! "# $%& '()*+,-./0123456789:;<=>?@ ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcdefghijklmnopqrstuvwxyz{|}
```

NONSB

```
▒_~ !"# $% & '()*+,-./0123456789:;<=>?@ ABCD EFGHIJKLMN OPQ RSTUV WXYZ[\]↑←abcdefghijklmnopqrstuvwxyz{|}
```

NONSBI

```
▒_~ ! "# $%& ' ()*+,-./0123456789: ; <=>?@ ABCDEFGHI JKLMN OPQRSTUVWXYZ[\]↑←abcdefghijkl mnopqrstuvwxyz{|}
```

NONM

```
▒_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcdefghijklmno
pqrstuvwxyz{|}
```

NONMI

```
▒_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑← abcdefghijklmn
opqrstuvwxyz{|}
```

NONMB

```
▒_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcdefghijklmn
opqrstuvwxyz{|}
```

Figure 16

NONS

```
▓_~!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcdefghijklmnopqrstuvwxyz{|}
```

NONMBI

```
▓_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑← abcdefghi
jklmnopqrstuvwxyz{|}
```

NONL

```
▓_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcdefghij
klmnopqrstuvwxyz{|}
```

NONLI

```
▓_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcdefgh
ijklmnopqrstuvwxyz{|}
```

NONLB

```
▓_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcde
fghijklmnopqrstuvwxyz{|}
```

NONLBI

```
▓_~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←abcd
efghijklmnopqrstuvwxyz{|}
```

Figure 17

GRFX10

↓┼^\ ├┬┤ϒδ∫±⊕∞☜▌⊓ //⊥X‖↔_→|≠⟋⟍ ≡v !"•┘└\/ ()*+,-./0123456789:;<=>?⇒ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←` abcdefg
hijklmnopqrstuvwxyz{|◊}^

GRFX14

↓┼^\ ├┬┤ϒδ∫±⊕∞☜▌⊓ // ⊥X‖↔_→|≠⟋⟍ ≡v !"•┘└\/ ()*+,-./0123456789:;<=>?⇒ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←` abcd
efghijklmnopqrstuvwxyz{|◊}^

Figure 18

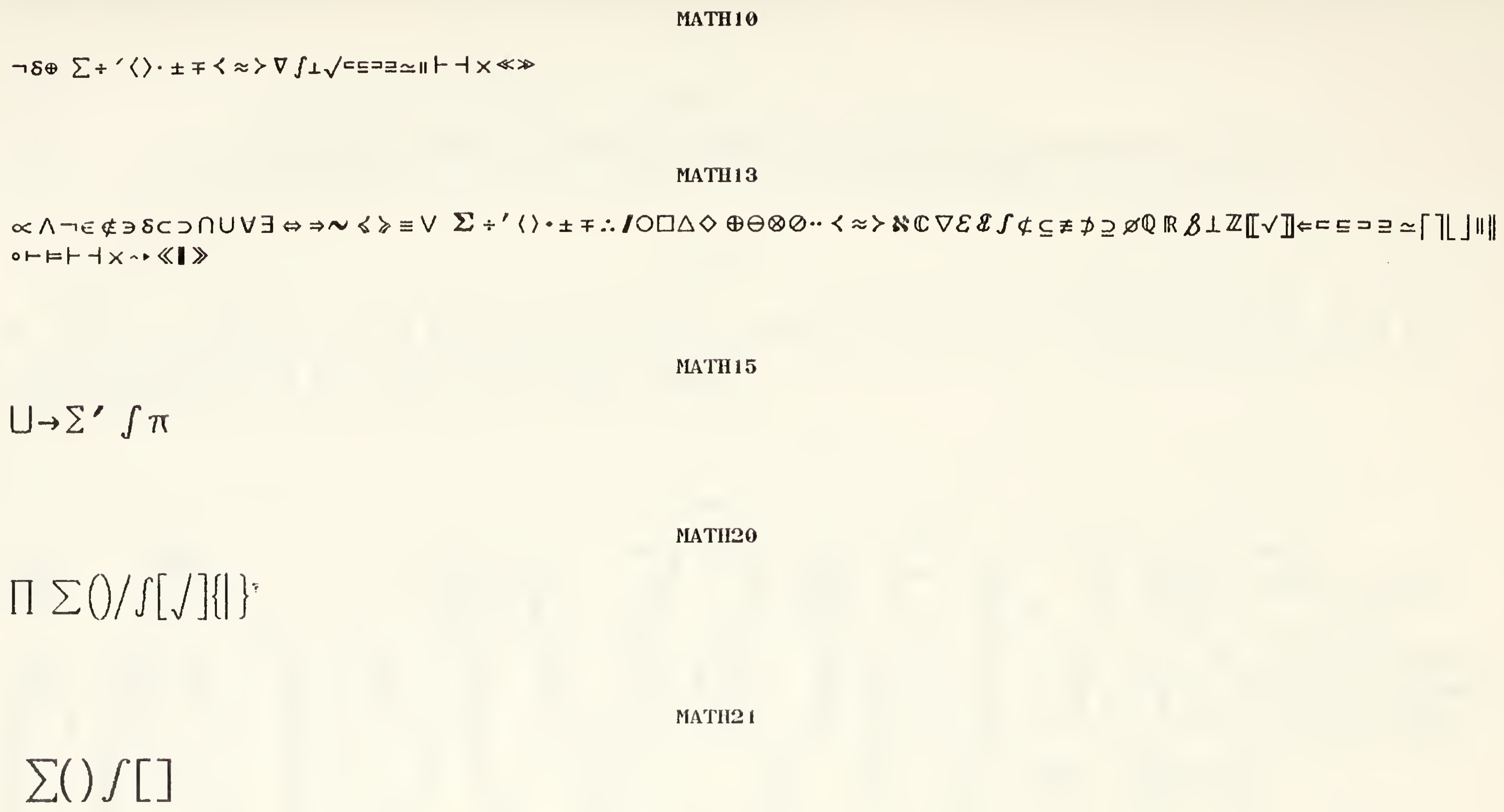


Figure 19

SAIL10

↓αβ∧¬∊πλ ↟↑ ∞∂⊂⊃∩∪∀∃⊗↔_→~≠≤≥≡∨ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGH
IJKLMNOPQRSTUVWXYZ[\]↑←'abcdefghijklmnopqrstuvwxyz{|}}_

SHD15

⇒⇔ !"#'+,.0123456789:;ABCDEFGHIJKLMNOPQRSTUVWXYZ[]⇐'

SIGN22

⇒ !'*,-./Ø123456789:;=ABCDEFGHIJKLMN
OPQRSTUVWXYZ←

SIGN41

⇒ !'*,-./Ø123456789:;=
ABCDEFGHIJKLMNOP
QRSTUVWXYZ←

Figure 20

What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
What is written without effort is in general read without pleasure. -Johnson
WHAT IS WRITTEN WITHOUT EFFORT IS IN GENERAL READ WITHOUT PLE
WHAT IS WRITTEN WITHOUT EFFORT IS
WHAT IS WRITTEN

Figure 21

APPENDIX B PROGRAM LISTINGS

This appendix contains the program listings, the source codes, for the programs described in the body of the thesis. Each listing is preceded by a one page description to avoid having to refer to various chapters for general information. One of the advantages of the program language C [Ref. 10] is that, while not self-documenting, it has constructs which are very descriptive; however, where necessary, comments have been added. Subroutines within programs are generally listed in a standard manner. "Main" appears first and is followed by subroutines in order of decreasing prominence.

DESCRIPTION

```
listfont [-l] <filename>
```

Listfont process a font file of the Stanford format. It examines the header table, the font dimensions, and the ASCII description. In doing so, Listfont ignores wasted high order bits and interprets 18-bit PDP-10 halfwords as 16-bit PDP-11 full words. The font dimensions and description are displayed on the CRT screen. Listfont then processes each character definition, detecting and flagging discrepancies in character dimensions or character picture storage. An optional "-l" argument displays character dimensions and pictures to the CRT screen.

FILES

<filename> must be a Stanford formatted file which has been read into a UNIX file.

```
#define OK if(printflag)
//controls optional char dimension/picture listing
float bytc;
int charw[128];
int caddr[128];
int *y, *z;
char textbuf[480], hbuf[6];
int bitptr,b,fp,bleft;
int printflag 0;
int unused 0;
int raf 0;
int lkf 0;
char *d;

main(argc,argv)
    int argc; char **argv;     {
    int i,j;
    if (--argc == 0)     {
       printf("PARAMETERS ?");
       exit( );
    }
    if (argv[1][0] == '-') {// turn on printflag
       printflag = 1;
       fp = OPEN(argv[2],0);
    }
    else fp = OPEN(argv[1],0);
    ptrblk( ); //get hdr table
    charblk( ); //get font dimensions
    fontblk( ); //get font ascii description
    OK printf("3. Character Definitions:\n");
    OK pblnkln(2);
    j = 128 - unused;
    //process the 'j' characters in the font
    for(i=0; i<j; i++)
       chardefs( );
    pblnkln(1);
    //report kerning or dimension errors
    if(raf)printf("Raster widths != char widths...%d\n",raf);
    else printf("Raster width - char widths all equal\n");
    if (lkf) printf("Kerning occurs %d times\n",lkf);
    //raf should equal lkf
    else printf("No nonzero left kerns\n");
    //if this doesn't agree with a 'ls -l filename'
    //then there are extraneous bytes present
    printf("Total bytes processed= %f\n",bytc+1.);
    CLOSE(fp);
}

ptrblk( ) {//go thru hdr table, count chars in font
    int i;
    bytc = -1.;
    y = charw; z = caddr;
```

```
    for(i=0; i<128; i++)     {
        *y++ = gethw( );
        *z++ = gethw( );
    }
    for(i=0;i<128;i++) if (charw[i] == 0) unused++;
}

int gethw( ) {//3 PDP-10 bytes to 2 PDP-11 bytes
    int c,t;
    READ(fp,&c,1);
    READ(fp,&c,1);
    READ(fp,&t,1);
    bytc =+ 3.;
    return( (c << 6) | t );
}

bytes(x)
    int x; {//trash x bytes and bump up counter
    int i,t;
    for(i=1; i<=x; i++)     {
        READ(fp,&t,1);
        bytc =+ 1.;
    }
}

pblnkln(x)
    int x; {//print x blank lines
    int i;
    for (i=1; i<=x; i++)
         putchar('\n');
}

pblnk(x)
    int x; {//print x blanks
    int i;
    for(i=1; i<=x; i++)
         putchar(' ');
}

charblk( ) {//print font dimensions

    printf("1. Characteristics:\n");
    pblnkln(2);
    bytes(9);
    printf("    Overall height of font (pixels)=  %d\n",
         gethw());
    bytes(3);
    printf("    Width of widest character=        %d\n",
         gethw( ));
    bytes(3);
    printf("    Logical height above baseline=    %d\n",
         gethw( ));
    bytes(168);
    pblnkln(4);
}
```

```
fontblk( ) {//print font ascii description

   int i,n;
   char c;
   n = bleft = b = 0;
   bitptr = 2;
   while ((c = nextchar( )) != 0)
      textbuf[n++] = c;
   printf("2. Font Description:\n");
   pblnkln(3);
   for(i=0; i<n; i++) putchar(textbuf[i]);
   pblnkln(1);
   bytes(576 - b);
}

char nextchar( ) {//get next char in ascii description

   char temp;
   int i,j;
   temp =& 0;
   for(i=0; i < 7; i++)    {
      if (bitptr == 2 && bleft == 0)    {
         d = hbuf;
         READ(fp,hbuf,6);
         bytc =+ 6.;
         bleft = 6;
         b =+ 6;
      }
      switch (bitptr)    {

         case 2: temp = temp | (*d & 040)  >> 5; break;
         case 3: temp = temp | (*d & 020)  >> 4; break;
         case 4: temp = temp | (*d & 010)  >> 3; break;
         case 5: temp = temp | (*d & 004)  >> 2; break;
         case 6: temp = temp | (*d & 002)  >> 1; break;
         case 7: temp = temp | (*d & 001); break;

         default: printf("bitptr= %d\n",bitptr);
                  exit( );

      }
      if (i<6) temp =<<  1;
      if (++bitptr > 7 || (bleft == 1 && (bitptr-1) == 6))  {
         bitptr = 2;
         bleft =- 1;
         d++;
      }
   }
   return( temp  );
}

chardefs( ) {//process one character definition

   char i,oc,tp,l,t,rft;
   int defc,drc,rw,rk;
```

```
   READ(fp,&tp,1);
   tp = tp << 3;
   READ(fp,&oc,1);
   rw = tp | ((oc & 070) >> 3);
   READ(fp,&t,1);
   bytc =+ 3.;
   oc = (oc << 6) | t;
   OK printf("Octal code=     ");
   if ( oc<8 ) OK printf("00");
   else if ( oc<64 ) OK putchar('0');
   OK printf("%o",oc);
   OK pblnk(16);
   defc = gethw( )-2;
   if (defc <= 0) {//stop here, there is no picture
      OK printf("NONPRINTABLE\n");
      OK pblnkln(2);
      bytes(6);
      return;
   }
   OK printf("Character width=  %d\n", charw[oc]);
   rw = (rw == 0) ? charw[oc] : rw;
   //rw != cw -> better be kerning
   if (rw != charw[oc]) raf++;
   OK printf("Raster width=     %d",rw);
   OK pblnk(15);
   READ(fp,&l,1);
   l = (l << 3);
   READ(fp,&rft,1);
   l = l | ((rft & 070) >> 3);
   READ(fp,&t,1);
   rft = ( rft << 6 ) | t;
   bytc =+ 3.;
   OK printf("Left kern=         %d\n", l);
   rk = rw - (charw[oc] + l);
   //all dimensions better jive
   if (rk < 0) OK printf("FILERROR - ");
   OK printf("Right kern=      %d",rk);
   if (rk || l) lkf++;
   OK pblnk(17);
   OK printf("Rows from top=  %d\n", rft);
   drc = gethw( );
   OK printf("Data row count=  %d", drc);
   OK pblnk(16);
   OK printf("defc= %d\n",defc);
   //now walk thru picture definition
   rastrln(defc,drc,rw);
   OK pblnkln(2);
}

rastrln(defc,drc,rw)
   int defc,drc,rw; {//process char picture definition
   int i,j,l,numrw,m;
   char t;
   int buf[90];
   int pbuf[270];
```

```
    int *p,*q,*n;
    OK pblnkln(1);
    //how many PDP-10 bytes per raster line?
    l = ((rw-1)/36 + 1)*6;
    //how many raster lines each 6 bytes?
    m = (l == 6) ? 36/rw : 1;
    while(drc) {//while data rows are left
        p = buf;
        for(i=0; i < l; i++)     {
            READ(fp,&t,1);
            bytc =+ 1.;
            *p++ = ((t & 070) >> 3);
            *p++ = (t & 07);
        }
        q = n = pbuf;
        p = buf;
        for(i=0; i < 2*l; i++)     {
            *q++ = f(p);
            *q++ = s(p);
            *q++ = td(p++);
        }
        numrw = (drc < m) ? drc : m;
        for(i=0; i < numrw; i++)     {
          for(j=0; j < rw; j++)
             OK list(n++);
          OK printf("\n");
        }
        drc =- numrw;
        defc =- l/6;
    }
    //trash any extraneous bytes
    bytes(defc*6);
}

list(n)
    int *n; {//use f(p),s(p),td(p) to list picture
    if (*n == 0)  printf(" ");
    else  printf("1");
}

int f(p)
    int *p;   {
    switch (*p)     {

        case 0: case 1: case 2: case 3:
            return(0);

        case 4: case 5: case 6: case 7:
            return(1);
        default:  printf("help");
    }
}

int s(p)
    int *p;     {
```

```
   switch (*p)    {

      case 0: case 1: case 4: case 5:
         return(0);

      case 2: case 3: case 6: case 7:
         return(1);
      default:  printf("help1");

   }
}

int td(p)
   int *p;    {
   switch (*p)    {

      case 0: case 2: case 4: case 6:
         return(0);

      case 1: case 3: case 5: case 7:
         return(1);
      default:  printf("help2");

   }
}
```

DESCRIPTION

```
transfile <sf> <nf>  <sf> <nf> ... <sf> <nf>
```

Transfile takes pairs of arguments. It transforms the first argument of a pair, a Stanford font file, to a font file of the NPS format with the name of the second argument of the pair. Transfile exits if given an odd number of arguments or a nonexistent file. Transfile detects and corrects dimensioning errors, removes unused bytes, and displays the transformed file's size before preceding to the next pair of arguments or exiting.

FILES

<sf> must be a Stanford formatted file.

<nf> will be shortened to length zero if it already exists.

```
#define READWRITE 00666 //access mode for transformed file
int fpr,fpw;
int notused  0;
char *bytc; //bytc counter
int dead;
char ibuf[256];
char tbuf[25];
char pbuf[25];
int obuf[256];
int big, blkc;
int flag;
int charwptr[256];
char *p;
int bitptr;
char textbuf[480];
char *d;
int g,bleft;

/* transform font files from the Stanford format to
   the NPS format; correct errors as detected      */

main(argc,argv)
   int argc;
   char **argv;     {
   int i,k,j, fileptr;
   putchar('\n');
   printf("\nTransform files by pair...\n");
   printf("FILES:  ");
   for(i=1;i<argc;i++) printf("%s ",argv[i]);
   putchar('\n');
   fileptr = 1;
   if ((argc--)%2 != 1)
      printf(Incorrect number of arguments\n");

   /* by pairs, transform the 1st argument(Stanford file)
      to the 2nd argument(NPS file)........
      .......continue until pairs of args are exhausted */

   else while (argc)  {
      big = 0; bytc = 0; blkc = 0;
      if (cmpr((p=argv[fileptr]),(d="sign114"))) big = 1;
      //set 'big' for the big file
      fpr = open(argv[fileptr++],0);
      fpw = creat(argv[fileptr++],READWRITE);
      for(i=0; i < 256; i++)
         obuf[i] = 0;
      write(fpw,obuf,512); //write blank hdr table
      bump(512); //set the byte counter
      for(i=0; i < 256; i++)     {
         charwptr[i] = getval( );
         if (charwptr[i] == 0) notused =+ 1;
      }
```

```
      kill(9); dead = putsave( );
      kill(3); dead = outsave( );
      kill(3); dead = putsave( );
      kill(168);
      fontblk( ); //get ascii description
      k = 0;
      for(j=0;textbuf[j] != '\0';j++)
                       ;
      for(i=j;(textbuf[i]=argv[fileptr-1][k]) != '\0';i++)
         k++;
      //write ascii description
      for(i=0;textbuf[i] !='\0';i++)     {
         putchar(textbuf[i]);
         write(fpw,&textbuf[i],1);
         bump(1);
      }
      write(fpw,&textbuf[i],1);
      bump(1);
      putchar('\n');
      dead = 128 - notused/2;
      //process chars in font
      for(i=0; i < dead; i++)
         chardef( );
      //go back to head of file
      seek(fpw,0,0);
      for(i=0; i < 256; i++)
         obuf[i] = charwptr[i];
      //write out the hdr table
      write(fpw,obuf,512);
      /* close files, write out byte count (this should
         agree with a 'ls -l' on transformed file), dec-
         rement the argument counter by a pair (2)      */
      close(fpr);
      close(fpw);
      if(big)printf("Size of %s...%d blocks + %d bytes\n",
                        argv[fileptr-1],blkc,bytc);
      else printf("Size of %s...%d bytes\n",argv[fileptr-1],
                     bytc);
      outchar('\n'); outchar('\n');
      argc =-  2;
   }
}

int cmor(p1,p2)
   char *p1,*p2; { //rtn 1 if 1o=p2, 0 otherwise
   for( ; ; )     {
      if(*o1 != *p2++)return(0);
      if(*p1++ == '\0')return(1);
   }
}

bump(i)
   int i; { //bump blk,byte counts by i as required
   if (big)     {
      if (bytc+i >= 512)     {
```

```
            if (blkc < 255)     {
               blkc++;
               bytc = (bytc+i)%512;
            }
            else if (bytc+i > 65535)  {
               printf("file too big"); exit();
            }
            else bytc =+ i;
         }
         else bytc =+ i;
      }
      else bytc =+ i;
}

int getval( ) { //3 bytes to 2
   read(fpr,ibuf,3);
   obuf[0] = (( ibuf[0] & 017) << 12) |
             (( ibuf[1] & 077) << 6)  |
             (  ibuf[2] & 077);
   return(obuf[0]);
}

int outsave( ) { //3 to 2 and write them
   read(fpr,ibuf,3);
   obuf[0] = (( ibuf[0] & 017) << 12) |
             (( ibuf[1] & 077) << 6)  |
             ( ibuf[2] & 077);
   write(fpw,obuf,2);
   bump(2);
   return(obuf[0]);
}

kill(x)
   int x; { //trash x bytes
   read(fpr,ibuf,x);
}

int wordc( ) { //rtn the number of 6 byte words
               //to the character picture
   read(fpr,ibuf,3);
   obuf[0] = (( ibuf[0] & 017) << 12) |
             (( ibuf[1] & 077) << 6)  |
             ( ibuf[2] & 077);
   obuf[0] =- 2;
   return(obuf[0]);
}

int retrw(){ //get rw, write rw, cmpr rw to cw
             //if rw != cw, set flag to check lk
   flag = 0;
   read(fpr,ibuf,3);
   obuf[0] = ((ibuf[0]&077) << 3) | ((ibuf[1] & 070) >> 3);
   obuf[1] = (( ibuf[1] &07) << 6) | ( ibuf[2] & 077);
   obuf[0] = (obuf[0] == 0) ? charwptr[2*obuf[1]] : obuf[0];
   charwptr[2*obuf[1] + 1] = bytc;
```

```
    if (big) charwptr[2*obuf[1]] =| (blkc << 8);
    if (obuf[0] != (charwptr[2*obuf[1]] & 0377)) flag++;
    write(fpw,obuf,2);
    bump(2);
    return(obuf[0]);
}

split( ) { //get, write out lk and rft
    int t;
    read(fpr,ibuf,3);
    obuf[0] = ((ibuf[0]&077) << 3) | ((ibuf[1]&070) >> 3);
    obuf[1] = (( ibuf[1] & 07) << 6) | ( ibuf[2] & 077);
    //correct any errors
    if (!flag) obuf[0] = 0;
    write(fpw,obuf,4);
    bump(4);
}

char next(x)
    int x; { //rtn value of next x bits to pak
    char temp;
    int i;
    temp =& 0;
    for(i=0; i < x; i++)     {
        switch (bitptr)     {

            case 0: temp = temp | (*p & 0200) >> 7; break;
            case 1: temp = temp | (*p & 0100) >> 6; break;
            case 2: temp = temp | (*p & 040)  >> 5; break;
            case 3: temp = temp | (*p & 020)  >> 4; break;
            case 4: temp = temp | (*p & 010)  >> 3; break;
            case 5: temp = temp | (*p & 004)  >> 2; break;
            case 6: temp = temp | (*p & 002)  >> 1; break;
            case 7: temp = temp | (*p & 001); break;
            default: printf("bitptr= %d\n",bitptr);
                     exit( );

        }
        if ( (i+1) != x ) temp =<< 1;
        if (++bitptr > 7)     {
            bitptr = 0;
            p++;
        }
    }
    if (i < 8) temp =<< (8 - i);
    return( temp & 000377 );
}

pak(x)
    int x; { //pak 1 raster line into int array
    int i;
    for(i=0; i < 25; i++)
        obuf[i] =& 0;
    i = 0;
    while (x)     {
```

```
        pbuf[i++] = next( (x >= 8) ? 8 : x);
        x = (x >= 8) ? x-8 : 0;
    }
}

comprs(x)
    int x; { //cmps int array into bits
    int k,bitsl;
    char *i,*t;
    t = tbuf; i = ibuf;
    for(k=0;k<25;k++)  *t++ =& 0;
    t = tbuf;
    bitsl = 8;
    while (x)     {
        switch (bitsl)     {

            case 2: *t = *t | (*i & 060) >> 4;
                    t++;
                    *t = *t | (*i++ & 017) << 4;
                    x--; bitsl = 4;
                    break;

            case 4: *t = *t | (*i & 074) >> 2;
                    t++;
                    *t = *t | (*i++ & 003) << 6;
                    x--; bitsl = 6;
                    break;

            case 6: *t = *t | (*i++ & 077);
                    t++;
                    x--; bitsl = 8;
                    break;

            case 8: *t = *t | (*i++ & 077) << 2;
                    x--; bitsl = 2;
                    break;

            default: printf("bitsl= %d\n",bitsl);
                     exit( );

        }
    }
}

chardef( )  //process one char definition
    int i;
    int rw;      //raster width
    int count;  //# wds in definition
    int rwperwd;//raster lines per word
    rw = retrw();
    count = wordc( );
    split( );
    drc = putsave( );
    while (drc) { //while data rows are left
        p = tbuf;
```

```
        bitptr = 0;
        if (rw > 36)     {
            read(fpr,ibuf,(rw/36 + 1)*6);
            comprs((rw/36 + 1)*6);
            pak(rw);
            write(fpw,pbuf,(rw%8 == 0) ? rw/8 : rw/8 + 1);
            bump( (rw%8 == 0) ? rw/8 : rw/8 + 1 );
            drc =- 1;
            count =- rw/36 + 1;
        }
        else     {
            read(fpr,ibuf,6);
            comprs(6);
            rwperwd = (drc < 36/rw) ? drc : 36/rw;
            for(i=0; i < rwperwd; i++)     {
                pak(rw);
                write(fpw,pbuf,(rw%8 == 0) ? rw/8 : rw/8 + 1);
                bump( (rw%8 == 0) ? rw/8 : rw/8 + 1 );
            }
            drc =- rwperwd;
            count =- 1;
        }
    }
    //trash extraneous bytes
    kill(count*6);
}

fontblk( )  { //get ascii description
    int i,n;
    n = bleft  = 0;
    g = 0;
    bitptr = 2;
    while ((textbuf[n++] = nextchar( )) != '\0')
                                ;
    kill(576 - g);
}

char nextchar( ) { //get next ascii char of descrip
    char temp;
    int i,j;
    temp =& 0;
    for(i=0; i < 7; i++)     {
        if (bitptr == 2 && bleft == 0)     {
            d = ibuf;
            read(fpr,ibuf,6);
            bleft = 6;
            g =+ 6;
        }
        switch (bitptr)     {

            case 2: temp = temp | (*d & 040)  >> 5; break;
            case 3: temp = temp | (*d & 020)  >> 4; break;
            case 4: temp = temp | (*d & 010)  >> 3; break;
            case 5: temp = temp | (*d & 004)  >> 2; break;
            case 6: temp = temp | (*d & 002)  >> 1; break;
```

```
         case 7: temp = temp | (*d & 001); break;
         default: printf("bitptr= %d\n",bitptr);
                  exit( );
      }
      if (i<6) temp =<<  1;
      if (++bitptr > 7 || (bleft == 1 && (bitptr-1) == 6)) {
         bitptr = 2;
         bleft =- 1;
         d++;
      }
   }
   return( temp  );
}
```

DESCRIPTION

```
edf [-][<Hfn>] ¦ [-][<Hfn>][<num>] ¦ [<fn>]
```

Edf is an interactive font editor which provides the capability of creating and maintaining fonts. If given no arguments, Edf enters a create mode. A filename, if given is assumed to be the name of a digitized font file; otherwise, a leading "-" indicates that <Hfn> is a vector formatted font file that requires conversion to a digitized form before the editing function may proceed. If a point size is not specified as an optional third argument, a vector formatted font will be digitized at a 10 point size. The term "current character" (cc) is the pointer to any character position in a font. The character denoted by cc may or may not be in the character buffer at any specified time. A user's manual [Ref. 7] gives a complete description of Edf and its use. Briefly, the available commands are:

<number>	set cc to <number>
+¦-	increment¦decrement cc
a	add a character to the font at the cc
c s e	change lines s through e of the character at cc, prompting for each line
d s e	delete characters s through e from the font
e s e	edit lines s through e of the character at cc, prompting for each line
f	turn on/off a switch displaying dimensions of the character at cc
i s e fn	include characters s through e from font fn fn must be compatible; remembers fn
l s e	list lines s through e of the character at cc
n	display the font decription and a table reflecting the edit status of every character in the font
o	enter an interactive module to change any

font/character dimension or font description

q	quit, warn if changes have been made but not written out
sl s e r u d	shift lines s through e of the character at cc left, right, up, or down one pixel and list lines s through e
w fn	write out font to fn, then quit
<rubout> <break>	kill any command being executed without exiting the program

Edf prompts with the octal value of cc followed by a ">" and questions "?" any illegal commands. Commands to change cc may be prepended to any other command, and the effect is to change cc and then execute the command. Additionally, "f" may be prepended to any command. Numbers may be in decimal or octal (leading 0).

FILES

<fn> may be a full pathname; otherwise, "/.fonts.01/font/" is prepended to it. Digitized Hershey fonts are placed in a temporary file named "/.fonts.01/HFONT".

```
#define error return(1);

int readfp, writefp;  //file descriptors
int ptsize;           //Hershey font point size
int pid;              //Child process id
int freenode;         //ptr to next free node in llist
int infont;           //current character
int wrflag;           //initially, 0. incremented on
                      //any change to flag a quit without
                      //writing
int wr;               //flag to turn off displaying of
                      //diagnostics during file writing
int max;              //32677 used to denote base node
int ht, maxw, lht;    //font dimensions
int blkc; char *bytc; //block,byte counters
int edit;             //set to 1 when in edit mode
int delete;           //flag in checking for empty fontfiles
int tht, tmaxw, tlht; //temp font dimensions
int dim;              //char dim diplay control switch
int include;          //flag preventing access to llist
                      //during an include command
int rw, lk, rft;      //character dimensions
int bot, bytes, drc;  //    "           "
int s, e;             //command arguments
int in;               //1 if current character definition is
                      //in character buffer, 0 otherwise
int c, peekc;         //characters on the command line
int first, last;      //line ptrs in character buffer
int chmod;            //1 if char in  buffer was modified
int *n;               //integer pointer
                      //0, otherwise
int sgtty[3];         //buffer for gtty(II)
int savetty;          //terminal status
int onintr();         //address of interrupt trap
int *chardef, *p;     //character pointers
char cstat;           //holds status of char in char buffer
char des[80];         //holds font description
char ibuf[36];        //buffer for read(II)
char tbuf[4000];      //character buffer
int hdr[256];         //hdr table of edited/created font
int fhdr[256];        //temp hdr table during an include
struct node    {      //a node holds info on a single
                      //character stored on the llist
   int code;          //character code
   char *def;         //ptr to char definition
   int nsize;         //size of new definition
   char stat;         //status of modification
   struct node *next; //ptr to next node in llist
}  llist[129];
struct node *head;    //ptr to head of llist
struct node *avail;   //ptr to next free node
struct node *current; //ptr to node found in FIND
```

```
struct node *insert();//node returned by INSERT
char rfontfile[40];    //fontfile being included from
char wfontfile[40];    //file being written to
char sfontfile[40]{"/.fonts.01/font/"};
                       //pathname header of fontfile to
                       //be edited
char hfsize[5]{"10"}; //default pt size for Hershey font

main(argc,argv)
   int argc; char **argv;     {
   int i;
   if (argc > 1) {//arguments->edit mode
     if (argv[1][0] == '-') {//digitize Hershey font
       if (argc == 3) {//check any point size
           if ((ptsize = atoi(argv[2])) > 42)      {
              printf("point size exceeds 42");
              exit();
           }
           p = hfsize;
           for(i=0;(*p++ = argv[2][i]) != '\0';i++);
       }
       pid = fork() ;
       if ( pid != 0 )
         while ( pid != wait() ) ;
       else  //create process to digitize Hershey font
          execl("makehf","makehf",argv[1],hfsize,0);
        readfp = open("/.fonts.01/HFONT",0);
      }
      else if ( argv[1][0] == '/' ) {//full pathname
         readfp = open(argv[1],0);
      }
     else  {
         p = argv[1];
         for(i=16;(sfontfile[i] = *p++) != '\0';i++);
         readfp = open(sfontfile,0);
      }
   edit = 1;
   }
   init();
   signal(2,onintr);   //set interrupt trap
   while (1)     {
      setexit();
      printf("\n%3o> ",infont);
      peekc = (peekc == '\n') ? 0 : peekc;
      if (command())     {
         printf("?\n");
         if (peekc != '\n') while((c=getc()) != '\n') ;
      }
   }
}

init()   {
   int i;
   if (edit)     {
      if (readfp > 0) fonthdr();
```

```
        else    {
            printf("fontfile not found\n");
            exit();
        }
    }
    else {//create mode
        zhdr(hdr);
        printf("\nfont height ? ");
        while((ht=getnum()) < 0 || ht > 120)    {
            peekc = 0; printf("height ?  ");    }
        printf(" %d !\n",ht);
        peekc = 0;
        printf("maximum character width ?  ");
        while((maxw=getnum()) < 0 || maxw > 256)    {
            peekc = 0; printf("Maxwidth ?  ");    }
        printf(" %d !\n",maxw);
        peekc = 0;
        printf("logical height above baseline ?  ");
        while((lht=getnum()) < 0 || lht > ht)    {
            peekc = 0; printf("lht ?  ");    }
        printf(" %d !\n",lht);
        peekc = 0;
        printf("Type in any one-line");
        printf(" font description, if desired.\n");
        getname(des);
    }
    max = 32677; wrflag = 0;
    head->code = max;
    head->next = 0; chmod = 0;
    include = 1; freenode = 1;
    infont = 0; wr = 1;
    head = llist; avail = &llist[1];
}

zhdr(h)  //zero a hdr table
    int h[];    {
    register int i;
    n = h;
    for(i=0;i<256;i++) *n++;
}

int getc() {//return next char in command line
    if (peekc)    {
        c = peekc;
        peekc = 0;
    }
    else    {
        c = getchar();
        if (c != ' ') peekc = c;
    }
    return(c);
}

fonthdr() {//read hdr table and font dimensions
    int i; char t;
```

```
    read(readfp,hdr,512);
    read(readfp,&ht,2);
    printf("\nHeight %d  ",ht);
    if (ht > 120 || ht < 0)     {
        printf("too high"); exit(); }
    read(readfp,&maxw,2);
    printf("Maximum character width %d  ",maxw);
    if(maxw > 256 || maxw < 0)     {
        printf("too wide"); exit();}
    read(readfp,&lht,2);
    printf("Logical height %d\n",lht);
    if(lht > ht || lht < 0)     {
        printf("too high"); exit();}
    seek(readfp,518,0);
    p = des; t = 1;
    for(i=0; t != '\0';i++)     {
        read(readfp,&t,1);
        *p++ = t;
    }
}

int getnum() {//convert numeric string and rtrn value
    int i,base;
    i = 0;
    while((c = getc()) == ' ')    ;
    if (c >= '0' && c <= '9')     {
        base = (c-'0') ? 10 : 8;
        peekc = c;
        if (base == 10) while((c=getc()) >='0' && c<='9')   {
            peekc = 0;
            i = i*base + c - '0';
        }
        else while((c=getc()) >='0' && c<='7')    {
            peekc = 0;
            i = i*base + c - '0';
        }
        peekc = c;
        return(i);
    }
    else{//there was no numeric string
        peekc = 0;
        if (c == '+') return(-2);
        if (c == '-') return(-3);
        peekc = c; //c will be processed later
        return(-1);
    }
}

int command()     {
    /* Process the command line:
            update infont
            check command arguments
            execute command
        Any problems ? return a 1; otherwise, return a 0 */
    register i,j;
```

```
int temp, k, h, hb, lb;
switch(temp = getnum())     {

   case -2:    //increment infont
      if (chmod) putdef();
      infont++;
      in = 0; chmod = 0;
      break;

   case -3:    //decrement infont
      if (chmod) putdef();
      infont--;
      in = 0; chmod = 0;
      break;

   case -1: break;    //no change

   default:    //infont gets temp
      if (chmod) putdef();
      infont = temp;
      in = 0; chmod = 0;
      break;

}
if (infont < 0) infont = 127;    //check for wraparound
if (infont > 127) infont = 0;
while((c = getc()) == ' ')    ;
switch (c)     {

  case 'a':    //add a character
    instr(); c=getchar(); getdim(); p=tbuf;
    for(i=0;i<4000;i++) *p++ = 0 ;
     bytes = (rw%8 == 0) ? rw/8 : rw/8 + 1;
    in++; wrflag++; chmod++; break;

   case 'c':    //change lines s thru e
      if (gchardef(readfp))     {
         if (setse(ht)) error;
         sbase();
         for(i=s; i < e;i++)
            for(j=first; j < last+first; j++)
               tbuf[i*bytes+j] = 0;
         for(i=s; i <= e; i++)     {
            printf("%3d ",i);
            for(j=first; j < last+first;j++)
               tbuf[i*bytes+j] = getdef();
         }
         in++; cstat = 'm';
         wrflag++; chmod++;
      }
      else error;
      break;

   case 'd':    //delete char's s thru e
      if (setse(128)) error;
```

```
        cstat = 'd';
        for(infont=s; infont<=e;infont++)     {
            if(hdr[infont*2] == 0) continue;
            hdr[infont*2] = 0; putdef();
        }
        in = 0; wrflag++; break;

    case 'e':    //edit lines s thru e
        if(gchardef(readfp))     {
            if(setse(ht)) error;
            sbase();
            gtty(1,sgtty); savetty = sgtty[1];
            for(i=s; i<=e;i++)     {
                printf("\n%3d ",i);
                sgtty[1] =| 03; stty(1,sgtty);
                for(j=first;j<first+last;j++)
                    list("%c%c%c%c%c%c%c%c",tbuf[i*bytes+j]);
                sgtty[1] = savetty; stty(1,sgtty);
                printf("\n     ");
                for(j=first;j<first+last;j++)
                    tbuf[i*bytes+j] = getdef();
            } in++; wrflag++; chmod++; cstat = 'm';
        } else error; break;

    case 'f':    //switch char dimension flag
        dim =(dim) ? 0 : 1 ;
        break;

    case 'i':  //include char's s thru e from rfontfile
        if (setse(128)) error;
        getname(rfontfile);
        ppend(rfontfile,"/.fonts.01/font/");
        if((temp=open(rfontfile,0)) < 0)      {
            printf("cannot open %s",rfontfile); error;
        }
        cpy(hdr,fhdr); read(temp,hdr,512);
        read(temp,&tht,2); read(temp,&tmaxw,2);
        read(temp,&tlht,2);
        if (reject())     {
            printf("compatible ");
            cpy(fhdr,hdr); error;
        }
        in = include = 0;
        cstat = 'i'; wr = 0; drc = 1;
        for(infont=s; infont<=e; infont++)      {
            if (gchardef(temp)) putdef();
            else if(drc == 0) putdef();
        }
        close(temp); wr = 1;
        for(i=0;i<s;i++)     {
            hdr[i*2] = fhdr[i*2]; hdr[i*2+1] = fhdr[i*2+1];
        }
        for(i=e+1;i<128;i++)     {
            hdr[i*2] = fhdr[i*2]; hdr[i*2+1] = fhdr[i*2+1];
        }
```

```
          include = 1; wrflag++; break;

     case 'l':    //list lines s thru e
          if (gchardef(readfp))     {
               if (setse(ht)) error;
               sbase();
               for(i=s; i <= e;i++)     {
                    printf("\n%3d ",i);
                    for(j=first;j < last + first; j++)
                         list("%c%c%c%c%c%c%c%c",tbuf[i*bytes+j]);
               }
               in++;
          }
          else error;
          break;

     case 'n':    //display font description and table
          p = des;
          if(*p == '\0') printf("no description\n");
          else for(i=0;*p != '\0'; i++)
            putchar(*p++);
          putchar('\n');
          printf("     0     1     2     3     4");
          printf("    5     6     7");
          for(i=0; i<128;i++)     {
               if(i%8 == 0)     {
                    if (i == 0)printf("\n000");
                    else if (i < 0100)printf("\n0%o",i);
                    else printf("\n%o",i);
               }
          pstat(i);
          }
          printf("\n\n' ' undefined  'X' unmodified  ");
          printf("'I' included  ");
          printf("'D' deleted  'M' modified");
          break;

     case 'p':    //modify font/char dimensions
          instr(); c = getchar();
          getdim(); break;

     case 'q':    //quit, warn if not written
          if (wrflag)     {
               wrflag = 0;
               printf("write??");
               error;
          }
          exit();


     case 's':    //shift lines s thru e once
          if(gchardef(readfp))     {
               peekc=0; temp=getc();
               if (setse(ht))  error;
               sbase();
```

```
    switch (temp)    {

       case 'r':   //right
        for(i=s; i<=e; i++)    {
           lb = 0;
           for(j=first; j < first+last; j++)    {
              hb = lb; p = &tbuf[i*bytes+j];
              if (*p & 01) lb = 1; else lb = 0;
              *p =>> 1;
              if(hb) *p =| 0200;else *p =& 0177;
           }
        } break;

       case 'l':   //left
        for(i=s;i<=e;i++)    {
           hb = 0; lb = 0;
           for(j=first+last-1;j>=first;j--)    {
              p = &tbuf[i*bytes+j];
              if((*p&0200)>>7) hb = 1; else hb = 0;
              *p =<< 1; if(lb) *p =| 01; lb = hb;
           }
         } break;

        case 'u':   //up
         for(i=s; i<=e; i++)    {
            if(i == 0) continue;
            for(j=first; j<first+last;j++)
              tbuf[(i-1)*bytes+j] = tbuf[i*bytes+j];
          }
         for(j=first; j<first+last;j++)
            tbuf[e*bytes+j] = 0;
         break;

       case 'd':   //down
         for(i=e;i>=s;i--)   {
           if (i == ht-1 ) continue;
           for(j=first;j<first+last;j++)
             tbuf[(i+1)*bytes+j] = tbuf[i*bytes+j] ;
           }
         for(j=first;j<first+last;j++)
           tbuf[s*bytes+j] = 0 ;
         break;


       default: error;

    }  //list the shift
    for(i=s; i <= e; i++)    {
       printf("\n%3d ",i);
       for(j=first;j < first+last; j++)
          list("%c%c%c%c%c%c%c%c",tbuf[i*bytes+j]);
    }
    in++; wrflag++; chmod++; cstat = 'm';
 } else error; break;
```

```
case 'w':   //write to wfontfile and quit
   if (chmod) putdef(); wr = 0;
   getname(wfontfile);
   //no writing to file being edited
  if ( cmpr(wfontfile,sfontfile) ||
       cmpr(wfontfile,"HFONT") )   {
     printf("writing to existing file "); wr=1; error;
  }
   if((writefp=creat(wfontfile,0666)) < 0)     {
      printf("file "); error;
   }
   zhdr(fhdr);
   write(writefp,fhdr,512); //write blank hdr table
   write(writefp,&ht,2);
   write(writefp,&maxw,2);
   write(writefp,&lht,2);
   blkc = 1; bytc = 6; p = des;
   for(i=0; *p != '\0';i++)     {
      write(writefp,p++,1); bump(1);
   }
   write(writefp,p,1); bump(1); in = 0;
   for(infont=0; infont< 128; infont++)     {
      if (hdr[infont*2] == 0) continue; //no char here
      else if (find(infont)) {//get it from llist
         if (current->nsize == 0) continue;
         fhdr[infont*2]=(hdr[infont*2]&0377)|(blkc<<8);
         fhdr[infont*2+1] = bytc;
         write(writefp,current->def,current->nsize);
         bump(current->nsize);
         free(current->def);
      }
      else if (edit) {//get it from file
         i = gchardef(readfp);
         p = tbuf;
         fhdr[infont*2]=(hdr[infont*2]&0377)|(blkc<<8);
         fhdr[infont*2+1] = bytc;
         write(writefp,p,8); bump(8);
         p =+ bytes*rft + 8;
         write(writefp,p,bytes*drc);
         bump(bytes*drc);
      }
     else error;
   }
   seek(writefp,0,0);
   write(writefp,fhdr,512);
   delete = 1;
   //remove any empty fontfile
   for(i=0;i<256;i=+ 2) if(fhdr[i] > 0) delete = 0;
   if (delete){blkc = bytc = 0; unlink(wfontfile);}
   printf("%l\n",blkc*512+bytc);
   exit();


case '\n': break;        // sync
```

```
        default:
           printf("%c ",c);
           error;

     }
     return(0);
}

bump(i) //running count wfontfile size
         //in blocks and bytes
     int i;     {
     if (bytc+i >= 512)     {
        if ((blkc + (bytc+i)/512) < 255)     {
           blkc =+ (bytc+i)/512;
           bytc = (bytc+i)%512;
        }
        else if (bytc+i > 32768)  {
           printf("file too big"); exit();
        }
        else bytc =+ i;
     }
     else bytc =+ i;
}

int cmpr(p1,p2)  //rtn 1 if p1 != p2; otherwise, 0
     char *p1,*p2;     {
     for( ; ; )     {
        if (*p1 != *p2++) return(0);
        if (*p1++ == '\0') return(1);
     }
}

cpy(n1,n2)  //copy p1 to p2
     int *n1,*n2;     {
     int i;
     for(i=0;i<256;i++) *n2++ = *n1++;
}

ppend(p1,p2)  //prepend p2 to p1
     char p1[], p2[];     {
     char *b1, *b2, t[40];
     b1 = p1; b2 = t;
     while((*b2++ = *b1++) != '\0') ;
     b2 = p2; b1 = p1;
     while((*b1++ = *b2++) != '\0') ;
     b2 = t; b1--;
     while((*b1++ = *b2++) != '\0') ;
}

int reject() {  //rtn 1 if files are incompatible;ow, 0
     if(tht != ht || tlht != lht || tmaxw > maxw) return(1);
     else return(0);
}

onintr() {  //restore environ. reset int trap
```

```
    signal(2,onintr);
    if (savetty)     {
       sgtty[1] = savetty;
       savetty = 0;
       stty(1,sgtty);
       savetty = 0;
    }
    reset();
}

int gchardef(fp)
   /* Get the character definition for the current
   character, put it in the char buffer, expand
   blank rows, and display necessary diagnostics */
   int  fp;     {
   register i,j;
   register char *tp;
   if (in) return(1);  //it's already there, rtn 1
   if (find(infont) && include) { //it's on the llist
      if (current->stat == 'd')     {
         printf("deleted ");
         return(0);
      }
      tp = tbuf;
      chardef  = current->def;
      *tp++ = rw = *chardef++; rw =& 0377;
      rw =| (*tp++ = *chardef++) << 8;
      if (rw <= 0)     {
         printf("%o raster width %d ",infont,rw); return(0);
      }
      bytes = (rw%8 == 0) ? rw/8 : rw/8 +1;
      *tp++ = lk = *chardef++; lk =& 0377;
      lk =| (*tp++ = *chardef++) << 8;
      *tp++ = rft = *chardef++; rft =& 0377;
      rft =| (*tp++ = *chardef++) << 8;
      *tp++ = drc = *chardef++; drc =& 0377;
      drc =| (*tp++ = *chardef++) << 8;
      if(drc == 0)     {
         printf("printable ");
         return(0);
      }
      bot = ht - (drc + rft);
      for(i=0; i < rft; i++)
         for(j=0; j < bytes; j++) *tp++ = 0;
      for(i=0; i < drc; i++)
         for(j=0; j < bytes; j++) *tp++ = *chardef++;
      for(i=0; i < bot; i++)
         for(j=0; j < bytes; j++) *tp++ = 0;
      if (wr && dim) pchardim();
      return(1);
   }
   //get it from the file
   if (hdr[infont*2] == 0)     {
      printf("undefined "); return(0);
   }
```

```
    if ((j= (hdr[infont*2] & 0177400) >> 8) != 0)     {
        j =& 0377;
        seek(fp,j,3);
        seek(fp,hdr[infont*2+1],1);
    }
    else seek(fp,hdr[infont*2+1],0);
    read(fp,&rw,2);
    if (rw <= 0)     {
        printf("%o raster width %d ",infont,rw); return(0);
    }
    read(fp,&lk,2);
    read(fp,&rft,2);
    read(fp,&drc,2);
    if (drc == 0 && wr)     {
        printf("printable ");
        return(0);
    }
    bot = ht -(drc + rft);
    bytes = (rw%8 == 0) ? rw/8 : rw/8 + 1;
    tp = tbuf;
    *tp++ = rw & 0377;
    *tp++ = (rw & 0177400) >> 8;
    *tp++ = lk & 0377; *tp++ = (lk & 0177400) >> 8;
    *tp++ = rft & 0377; *tp++ = (rft & 0177400) >> 8;
    *tp++ = drc & 0377; *tp++ = (drc & 0177400) >> 8;
    for(i=0; i < rft; i++)
        for(j=0; j < bytes; j++) *tp++ = 0;
    for(i=0; i < drc; i++)     {
        read(fp,ibuf, bytes);
        for(j=0; j < bytes; j++) *tp++ = ibuf[j];
    }
    for(i=0; i < bot; i++)
        for(j=0; j < bytes; j++) *tp++ = 0;
    if (wr && dim) pchardim();
    return(1);
}

int setse(x)  //set command args s and e
    int x;     {
    peekc = 0;
    s = getnum();
    if (s < 0)     {
        s = 0; e = x-1;
        return(0);
    }
    e = getnum();
    if (e < 0) e = s;
    if (e < s) error;
    if((s >= x || e >= x) && x == 128) error;
    if((s > x || e > x) && x == ht) error;
    return(0);
}

list(fmt,byt)
//list byte, bit by bit, 0=>'.', 1=>'0'
```

```
    char *fmt, byt;     {
    printf(fmt,0200&byt?'0':'.',0100&byt?'0':'.',
               0040&byt?'0':'.',0020&byt?'0':'.',
               0010&byt?'0':'.',0004&byt?'0':'.',
               0002&byt?'0':'.',0001&byt?'0':'.');
}

int find(i)
//if current character is on llist, rtn 1 and
//current points to correct node; ow, rtn 0
    int i;    {
    register struct node *ptr;
    ptr = head;
    while (i > ptr->code )
        ptr = ptr->next;
    if (i == ptr->code)     {
        current = ptr;
        return(1);
    }
    else return(0);
}

getname(file)
//get name ending in '\0' and stick it in file
    char file[];     {
    while((c = getc()) == ' ')     ;
    if(c != '\n')     {
        p = file;
        do   {
            *p++ = c; peekc = 0;
        }   while((c = getc()) != '\n');
        *p = '\0';
    }
}

putdef()     {
//put definition in char buffer on llist
    if (find(infont)) lnode(current,infont);
    else     {
        lnode(insert(avail,infont),infont);
        if (freenode > 128)   {
            printf("overflow"); exit();
        }
        avail = &llist[++freenode];
    }
}

lnode(ptr,k)  //do the work for PUTDEF
    struct node *ptr; int k;     {
    register int i,j;register char *tp;
    int clear;
    ptr->code = k;
    if (cstat == 'd')     {
        ptr->stat = cstat;
        return;
```

```
    }  //count blank rows at top and bottom
    rft = bot = 0;
    i = 0; clear = 1;
    while(i < ht && clear)    {
        for(j=8; j < bytes + 8; j++)
            if (tbuf[i*bytes+j] != 0) clear = '\0';
        if (clear) rft = i+1;
        i++;
    }
    if (i < ht)    {
        i = ht-1; clear = 1;
        while(i > 0 && clear)    {
            for(j=8; j < bytes + 8; j++)
                if (tbuf[i*bytes+j] != 0) clear = '\0';
            if (clear) bot = ht-i;
            i--;
        }
    }
    drc = (drc) ? ht -(rft+bot) : 0;
    if(drc == 0) rft = lk = 0;
    tp  = ptr->def = alloc(bytes*drc+8);
    *tp++ = rw & 0377; *tp++ = (rw & 0177400) >> 8;
    *tp++ = lk & 0377; *tp++ = (lk & 0177400) >> 8;
    *tp++ = rft & 0377; *tp++ = (rft & 0177400) >> 8;
    *tp++ = drc & 0377; *tp++ = (drc & 0177400) >> 8;
    for(i=rft; i < rft+drc;i++)    {
        for(j=8; j < bytes + 8; j++)
            *tp++ = tbuf[i*bytes+j];
    }
    ptr->nsize = 8+drc*bytes;
    ptr->stat = cstat;
}

struct node *insert(a,i)
//rtn a node for PUTDEF to use
    struct node *a; int i;    {
    register struct node *ptr,*temp;
    temp = ptr = head;
    while( i > ptr->code )    {
        temp = ptr;
        ptr = ptr->next;
    }
    if (ptr == head)    {
        a->next = head;
        head = a;
    }
    else    {
        a->next = temp->next;
        temp->next = a;
    }
    a->stat = a->def = a->nsize = 0;
    return(a);
}

sbase() { //set horizontal starting point for char def
```

```
    first = 8; last = bytes;  //normal char, default
    if (bytes > 9) { //too wide, get a starting pt
       printf("\ntoo wide...starting where ?");
       peekc = 0;
       while((last = getnum()) < 0 || last >= rw)     {
          peekc = 0; printf("where ?");     }
       peekc = 0;
       last = (last == 0) ? 1 :  last/8 + 1;
       first = first + last-1;
       last = ((bytes+8-first) > 9) ? 9 : bytes+8-first;
    }
}

getdef() {  //get one byte of a definition
    int mask,i,j;
    peekc = 0;
    while((c = getc()) != '0' && c != '.')     ;
    peekc = c;
    i = j = 0;
    mask = 0400;
    while((j++ < 8) && ((c=getc()) == '0' || c == '.'))    {
       peekc = 0;
       if ((mask = mask>>1) && c == '0')
          i =| mask;
    }
    return(i);
}

pstat(i)  //print char status for edit table
    int i;     {
    if (find(i))     {
       switch(current->stat)     {

          case 'd': printf("  D  ");break;
          case 'i': printf("  I  ");break;
          case 'm': printf("  M  ");break;

       }
    }
    else if (hdr[i*2] == 0) printf("      ");
    else printf("  X  ");
}

pchardim() {  //display char dimensions
    int i;
    if((i = hdr[infont*2] & 0377) == 0)   {
       printf("undefined"); return;
    }
    printf("rw %d  cw %d  ",rw,i);
    if (rw == i) printf("lk %d  rk %d",lk,lk);
    else if (lk)    {
       if (lk+i == rw)printf("lk %d  rk %d",lk,0);
       else printf("lk %d  rk %d",lk,rw -i-lk);
    }
    else printf("lk %d  rk %d",lk,rw-i);
```

```
	printf("  ht %d  lht %d  ",ht,lht);
	printf("rft %d  drc %d\n",rft,drc);
}

getdim()	{
	/* Look for a number and/or name. Take both as
	a request, rejecting invalid requests with a '?'
	Quit on 't' and return to the main command loop */
	int i,j, font; char name[20];
	j = hdr[infont*2]&0377; font = 0;
	while (1)	{
		peekc = 0; printf("\n%3o-> ",infont);
		i = getnum(); getname(name);
		if(cmpr(name,"t")) break;
		if(cmpr(name,"i")) instr();
		else if(cmpr(name,"infont"))	{
			infont = i; i = gchardef(readfp);
		} else if(cmpr(name,"d"))	{
			printf("%s\n",des);
			peekc = 0; getname(des);
		} else if(cmpr(name,"p")) pchardim();
		else if(cmpr(name,"f"))
			printf("ht %d maxw %d lht %d\n",ht,maxw,lht);
		else if(cmpr(name,"ht"))	{
			if(i >= lht){ ht = i; wrflag++; }
			else printf("\n? ");
		} else if(cmpr(name,"lht"))	{
			if(i <= ht){ lht = i; wrflag++; }
			else printf("\n? ");
		} else if(cmpr(name,"maxw"))	{
			if(i < 0 || i > 256) {maxw = i; wrflag++; }
			else printf("\n? ");
		} else if(cmpr(name,"cw"))	{
			if(gchardef(readfp))	{
				if(i <= rw)	{
					hdr[infont*2] =& 0177400;
					hdr[infont*2] =| i & 0377;
					lk = rw-i; font = 1;
				} else printf("\n? ");
			} else printf(" cw now %d\n",(hdr[infont]=i));
		} else if(cmpr(name,"rw"))	{
			if (gchardef(readfp))	{
				if(i <= maxw)	{
					rw = i; font = 1;
					if(rw < j)	{
						hdr[infont*2] =& 0177400;
						hdr[infont*2] =| i & 0377;
						lk  = 0; font = 1;
					}
				} else printf("\n? ");
			} else printf(" rw now %d\n",(rw = i));
		} else if(cmpr(name,"lk"))	{
			if(gchardef(readfp))	{
				if(rw == j)	{
					if(i == 0) {lk = i; font = 1; }
```

```
                    else printf("\n? ");
                } else if(i <= rw-j) { lk = i; font = 1; }
                else printf("\n? ");
            } else printf(" lk now %d\n",(lk = i));
        } else if(cmpr(name,"rk"))    {
            if(gchardef(readfp))    {
                if(rw == j)    {
                    if (i == 0) ;  else printf("\n? ");
                } else if(i <= rw-j)    {
                    if(i+lk == rw-j) ;
                    else { lk = rw-i; font = 1; }
                } else printf("\n? ");
            } else printf("\n? ");
        } else printf("\n? ");
    }
    if (font)    {
        wrflag++;cstat = 'm';putdef();in = 0;
    }
}

instr() {  //display instructions for GETDIM
    printf("Modifiable FONT dimensions are:\n");
    printf("height- 'ht'  max character width- 'maxw'");
    printf("  logical height- 'lht'\n\n");
    printf("Modifiable CHARACTER dimensions are:\n");
    printf("raster width- 'rw'  character width- 'cw'");
    printf("  left kern- 'lk'  right kern 'rk'\n\n");
    printf("Type 'i' for instructions, 'p' for ");
    printf("dimensions of character in buffer.\n");
    printf("To move to another character, update ");
    printf("'infont'.\n");
    printf("\nGet font dimensions with 'f'. ");
    printf("Modify font name with 'd'. If you're adding");
    printf("a\n character, make changes in this order only:");
    printf(" 'rw', 'lk' , then 'cw'.\n");
    printf("\nImpossible modifications are rejected....");
    printf("some example inputs might be\n");
    printf(" '22 lht', '063 infont', 'i', or '0 lk'\n\n");
    printf("You'll be prompted with a '->'. ");
    printf("When you are finished, type 't'... \n\n");
}
```

DESCRIPTION

```
prfont [-<number>] <fn> <fn> ... <fn>
```

Prfont takes font names or full pathnames as arguments. For each argument, Prfont displays the font, setting the characters in the character code collating sequence. Character positions are set and appear as they would if used in documents. The fonts are displayed in a 9 inch horizontal field which may be adjusted by an optional leading argument, a decimal number between 1 and 264. The default field width (9 inches) is 216 bytes.

FILES

<fn> must be a digitized file.

```
#define SPACE 1              // one 1/4 inch vertical space
#define TOP 230              // top margin
#define PAGEHT 14*100
int roww, rows;
int linecount PAGEHT;
int pagewth;
int prdev, pldev, infont;
int ht, maxw, lht, fp;
int head, tail, nodeptr;
int zero[1], hdr[256];
char *lp, *p;
char ff 014; char nl 012;
char header[40] {"/.fonts.01/font/"} ;
char prbuf[132], plbuf[264];
struct cnode    {
   int cc;                  //char code
   char *optr;              //->1st raster line
   char *lptr;              //-> next raster line
   int rw;                  //raster line width
   int bytes;               //bytes per raster line
   int lk;                  //left kern
   int rft;                 //rows from top
   int drc;                 //data row count
} clist[128];
struct cnode *a;
struct cnode *fset[128];

main(argc, argv)
   int argc; char **argv;     {
   register int i, argptr;
   char go;
   argptr = 1;
   if((prdev=open("/dev/spp",1)) < 0)     {
      printf("cannot open printer");exit();}
   if((pldev=open("/dev/rvp",1)) < 0)     {
      printf("cannot open plotter");exit();}
   if (argv[1][0] == '-') {//reset pagewth
      pagewth = atoi( &argv[1][1] ); go = 1;   }
   else  { pagewth = 216; go = 0; }
   init();
   while(--argc != go) {//process all files
      p = argv[argptr+go];
      if ( *p == '/' ) { //full pathname
         if ( (fp=open(argv[argptr+go],0)) < 0) {
            printf("cannot open %s",argv[argptr+go]);
            exit();  }
         printf("%s opened....",argv[argptr+go]);
         }
      else { //prepend /fonts.01/font
         for(i=16;(header[i]= *p++) != '\0';i++) ;
         if((fp=open(header,0)) < 0)     {
           printf("cannot open %s",header);exit();}
```

```
                printf("%s opened.....",header);
                }
            infont = head = tail = nodeptr = roww = 0;
            read(fp,hdr,512); read(fp,&ht,2);
            read(fp,&maxw,2); read(fp,&lht,2);
            check(); //check for bad font file
           if ( ht <= 82 ) {
            //set vert spacing
             if (ht <= 40) rows = 2 ;
             else rows = 3 ;
             }
           else rows = 4 ;
            //pgbk if font display won't fit
            if(nroom(rows*ht + 40)) pagebreak();
            p = prbuf; for(i=0;i<60;i++) *p++ = ' ';
            for(i=0;(*p++ = argv[argptr+oo][i]) != '\0';i++);
            *p = nl;
            //center, write font name
            write(prdev,prbuf,i+62);
            for(i=0;i<25;i++) write(pldev,zero,2);
            linecount =+ 25;
            while (1)    {
                getrow();
                putrow();
                if(infont > 127) break;
            }
            close(fp); printf("closed\n"); argptr++;
            //if need be, pgbk
            if(nroom(SPACE*2)) pagebreak();
            else space(SPACE*2);
        }
        exit();
}

init()     {
        register int i;
        for(i=0;i<128;i++) fset[i] = &clist[i];
}

pagebreak() { //page eject
        int i;
        char err;
        err = cvers(pldev,020);
        if ( err == -1 ) {
        printf("invalid filedes in pagebreak\n");
        exit();
        }
        for (i=0;i<TOP;i++) write(pldev,zero,2);
        linecount = TOP;
}

getrow() { //get a row of chars to plot
        if(tail)     {
            roww = fset[++tail]->bytes;
            head = tail++;
```

```
    }
    while (1)    {
        if(getdef())    {
            if(roww + fset[tail]->bytes <= pagewth)
                roww =+ fset[tail]->bytes;
            else {tail--; ++infont; break;}
            if (++infont > 127) break;
            tail++;
        }
        else if(++infont > 127) break;
    }
}

putrow()  { //plot the row of characters
    register int h,i,l; int t;
    struct cnode *ptr;
    for(h=0; h < ht; h++)    {
        p = &plbuf[24];
        ptr = fset[(t = head)];
        for(l=head;l<=tail;l++)    {
            if(ptr->drc)    {
                if(h >= ptr->rft && h < ptr->rft+ptr->drc)    {
                    //lp-> next raster line
                    lp = ptr->lptr;
                    //do it by bytes
                    for(i=0;i<ptr->bytes;i++)
                        *p++ = *lp++;
                    //update lptr for next pass
                    ptr->lptr =+ ptr->bytes;
                }
                //blank line
                else for(i=0;i<ptr->bytes;i++) *p++ = 0;
            }
            //blank character
            else for(i=0;i<ptr->bytes;i++) *p++ = 0;
            ptr = fset[++t];
        }
        //plot 1 raster line of row of characters
        write(pldev,plbuf,roof(roww+24));
    }
    //row plotted, plot some white space
    for(h=0;h<5;h++) write(pldev,zero,2);
    linecount =+ ht+5;
    //free bytes in reverse order
    for(i=tail;i>=head;i--)
        if(fset[i]->optr)
            free(fset[i]->optr);
}

int getdef()    {
    int blkc,bytc; register i;
    if(hdr[infont*2])    {
        blkc = (hdr[infont*2]&0177400) >> 8;
        blkc =& 0377;
        bytc = hdr[infont*2+1];
```

```
        if(blkc) { //ptr is in blks and bytes
           seek(fp,blkc,3); seek(fp,bytc,1); }
        else seek(fp,bytc,0);
        getnode();
        a->cc = infont; read(fp,&a->rw,2);
        read(fp,&a->lk,2); read(fp,&a->rft,2);
        read(fp,&a->drc,2);
        a->bytes = (a->rw%8 == 0) ? a->rw/8 : a->rw/8+1;
        if(fcheck()) { //check for bad char dimensions
           if(a->drc) { //need bytes?, call alloc
              if((i=a->optr=a->lptr=alloc(a->drc*a->bytes))<0){
                 printf("\nout of memory...");
                 printf("use a smaller pagewidth\n");
                 exit();  }
              read(fp,a->lptr,a->drc*a->bytes);
        }
        return(1);
        }
     }
     return(0);
}

getnode()     {
     if(nodeptr > 127)     {
        printf("overflow"); exit();}
     a = fset[nodeptr++];
     a->optr = a->lptr = 0;
}

int roof(x)
     int x; { //send plotter even # bytes only
     if(x%2 == 0) return(x);
     //for some reason 264 bytes crashes program
     if(x == 263) return(262);
     *p = 0; return(++x);
}

space(x)
     int x; { //plot x 1/4 inches space
     int i;
     for(i=0;i<x*50;i++) write(pldev,zero,2);
     linecount =+ x*50;
}

check() { //print then exit on bad file
     if(ht < 0 || maxw < 0 || lht < 0 ||
        ht > 256 || maxw > 256 || lht > ht)    {
           printf("bad file"); exit();
     }
}

int nroom(x)
     int x; { //rtn 1 there are not x plot lines
             //left before bottom; otherwise, 0
     if(linecount + x > PAGEHT) return(1);
```

```
   else return(0);
}

fcheck() { //if bad chardef, rtn 0 to skip it
            //otherwise; rtn 1.
   if ( (a->rw<0 || a->rw>255) || (a->rft<0 || a->rft>255)
     || (a->lk<0 || a->lk>255) || (a->drc<0 || a->drc>255)
        )  {
      printf("\ninvalid value for character '%c'\n",infont);
      printf("rw %d\trft %d\tlk %d\tdrc %d\n",a->rw,
             a->rft,a->lk,a->drc);
      return(0);
      }
   else return(1);
}
```

DESCRIPTION

```
signmkr <fn>
```

Signmkr reads lines from <fn> and performs limited text processing. It sets the text in <fn> in the selected fonts. Reference 7 provides detailed instructions for its use; however, a brief description of available commands is listed below:

ESCf<fontname>	Change fonts
ESCc <text>	Center <text>
ESCo<character code>	Set the character indicated by the code
ESCpp	paragraph
ESCpg	pagebreak

FILES

<fn> is a text file interspersed with any of the above commands.

```
#define TOP 230          // top margin
#define PAGEHT 14*100
int roww;
int sl 0;
int pagewth 216;
int linecount PAGEHT;
int pldev, infont, in, base;
int ht, maxw, lht, fp, ip, r;
int nodeptr, openbits;
int zero[32], hdr[256];
char *lp, *p, *t, *n, *pl;
char esc 033; char blank 040; int c;
char header[40] {"/.fonts.01/font/"} ;
char pbuf[90], tbuf[90], plbuf[264];
char fmark[128];
char fontname[20], ochar[10];
struct cnode   {
   int cc;               //character code
   char *optr;           //->1st raster line
   int rw;               //raster line width
   int bytes;            //bytes per raster line
   int lk;               //left kern
   int rft;              //rows from top
   int drc;              //data row count
} clist[128];
struct cnode *a, *ptr;
struct cnode *fchar[128];



main(argc, argv)
   int argc; char **argv;    {
   if (argc < 2) exit();
   else if ((ip=open(argv[1],0)) < 0)    {
      printf("cannot open %s",argv[1]); exit();
   }
   init();
   while (getln()) putln();
   printf("closed\n"); exit();
}

init()    {
   register int i;
   if((pldev=open("/dev/rvp",1)) < 0)    {
      printf("cannot open plotter"); exit();
   }
   for(i=0;i<128;i++) fchar[i] = 0;
   n = fmark; for(i=0;i<128;i++) *n++ = -1;
   fp = 0; cfont("SAIL10"); //default font
}

int getln() { //rtn 1 if there's a line to
```

```
                //be plotted;otherwise, 0
    char k;
    t = tbuf;
    k = 0;
    while ( ((*t = getch()) != '\n') &&
              (*t != '\0') )   {
      if ( k++ == 89 ) { *t = '\n'; break; }
      t++;
      }
    if ( *t == '\0' ) return(0);
    else return(1);
}

putln()  { //plot as much as can fit in PAGEWTH
    register int h,i;
    roww = 0; pagewth = 216;
    if ( sl == 0) sl = 24;
    t = tbuf; p = pbuf;
    while (*t != '\n')     {
        if (*t == esc) { if (eschar()) break; }
        if (filchar()) break;
    }
    *p = '\n';
    if (t == tbuf) return; //null line in input file
    //check for room
    if (nroom(ht+(ht/10+1))) pagebreak();
    for(h=0;h<ht;h++)     {
        pl = &plbuf[sl]; *pl = 0; openbits = 8;
        ptr = fchar[*(p = pbuf)];
        while (*p != '\n')     {
            r = ptr->rw;
            if (ptr->drc)     {
                if(h >= ptr->rft && h < ptr->rft+ptr->drc)     {
                    i = h - ptr->rft;
                    lp = ptr->pptr + i*ptr->bytes;
                    while(r > 0)     {
                        shift(); r =- 8; }
                } else     {
                    lp = zero;
                    while(r > 0)     {
                        shift(); r =- 8;}
                }
            } else     {
                lp = zero;
                while(r > 0)     {
                    shift(); r =- 8;}
            }
          ptr = fchar[*++p];
        }
        //plot one row raster line
        write(pldev,plbuf,roof(roww+sl*8));
    }
    //plot some white space
    for(h=0;h < ht/10+1;h++)
        write(pldev,zero,2);
```

```
    linecount =+ ht+(ht/10+1);
    sl = 0;
}

eschar()  { //esc- special characters
    int i, hi, space;
    char tt, *tb, *te;
    if (t == tbuf)    {
       if ((c = *++t) == 'f') { //font change
          n = fontname; t++;
          while ( (*n = *t++) != ' ' && *n != '\n' )
            n++;
          tt = *n;
          *n = '\0'; cfont(fontname);
          if ( (tt == '\n') || (*t == '\n'))   {
             t = tbuf; return(1);  }
       } else if (c == 's') { //need space
          n = ochar; t++;
          base = (*t == '0') ? 8 : 10 ;
          while (num(*n = *t)) {
            n++; t++;  }
          *n = '\0';
          hi = oct(ochar) * ht ;
          if (nroom(hi))   {
             pagebreak(); t = tbuf; return(1); }
          for (i=0;i<hi;i++)
             write(pldev,zero,2);
          linecount =+ hi ;
          t = tbuf; return(1);
       } else if (c == 'o'){ //no ascii equivalent
          n = ochar; t++;
          base = (*t == '0') ? 8 : 10;
          while (num((*n = *t)) )   {
            n++; t++;  }
          *n = '\0'; t--;
          *t = ((i = oct(ochar)) > -1 && i < 128 ) ? i
                                                : blank;
       } else if (c == 'c') { //center this line
          while (*++t == ' ') ;
          tb = t ;
          while (*++t != '\n') ;
          while (*--t == ' ') ;
          te = t; space = 0;
          for(t=tb; t<=te; t++)      {
            if (hdr[*t*2])
               space =+ hdr[*t*2] & 0377;
            else if (hdr[040*2])      {
               space =+ hdr[040*2] & 0377;
               *t = 040;
            } else     {
               printf("input error-- ");
               printf("\tundefined character...%c\n",*t);
               flushh();
            }
         }
```

```
        space = (space%8 == 0) ? space/8 : space/8+1;
        sl = 132 - space/2;
        if (sl < 24)    {
          printf("input error-- ");
          printf("\ttoo many characters to center\n");
          flushh();
       }
       for(i=0;i<sl;i++) plbuf[i] = 0;
       t = tb;
      } else if (c == 'p')  {
         if ( (c = *++t) == 'g')  {//pgbreak
            pagebreak(); t = tbuf; return(1);  }
         else if (c == 'p')  {//paragraph
                for(i=0;i<ht;i++)
                   write(pldev,zero,2);
                sl = 24 + (24 * ht/120);
                pagewth = pagewth - (24 * ht/120);
                t = tbuf; return(1);
                }
              else  {
                printf("invalid character folowing ");
                printf("'ESCp'..");
                exit();
                }
      } else      {
         printf("input error- ");
         printf("\tinvalid escape character... %c",c);
         flushh();
      }
   } else if ((c = *++t) == 'o') { //no ascii equiv
      n = ochar; t++;
      base = (*t == '0') ? 8 : 10 ;
      while (num((*n = *t)) )  {
         n++; t++; }
      *n = '\0'; t--;
      *t = ((i=oct(ochar)) > -1 && i < 128) ? i
                                      : blank;
   } else if (c == 'f') {//no font chg allowed here
      printf("change fonts at line head only ");
      flushh();
   } else     {
      printf("input error- ");
      printf("\tinvalid escape character ( %c )\n",c);
      printf("\tembedded within text...\n");
      flushh();
   }
   return(0);
}

int filchar()    { //move chars from tbuf to pbuf until
                   //PAGEWTH exceeded, replace nonexistent
                   //chars with blank; ow, exit
   register int i;
   infont = *t;
   if (hdr[infont*2])    {
```

```
        if (fchar[infont] == 0)    {
            getdef();
            if (roww+a->rw <= pagewth*8)
                roww =+ a->rw;
            else {*p = '\n'; return(1);}
        } else if (roww+fchar[infont]->rw <= pagewth*8)
                roww =+ fchar[infont]->rw;
        else {*p = '\n'; return(1);}
    } else if (hdr[(infont=blank)*2])    {
        *t = blank;
        if (fchar[infont] == 0)    {
            getdef();
            if (roww+a->rw <= pagewth*8)
                roww =+ a->rw;
            else {*p = '\n'; return(1);}
        } else if (roww+fchar[infont]->rw <= pagewth*8)
                roww =+ fchar[infont]->rw;
        else {*p = '\n'; return(1);}
    } else    {
        printf("character '%3o' not defined in %s",*t,
                header);
        flushh();
    }
    *p++ = *t++;
    return(0);
}

cfont(q)
    char *q; { //q points to new font name
    register int i;
    if (fp)   {
        printf("closed\n"); close(fp);
    }
    for(i=16;(header[i] = *q++) != '\0';i++)   ;
    if((fp=open(header,0)) < 0)    {
        printf("cannot open %s",header); exit();
    }
    printf("%s opened....",header);
    dealloc(nodeptr); nodeptr = 0;
    for(i=0;i<128;i++) fchar[i] = 0;
    read(fp,hdr,512); read(fp,&ht,2);
    read(fp,&maxw,2); read(fp,&lht,2);
    if(check())    {
        printf("%s bad font file",header);
        exit();
    }
}

dealloc(x)
    int x;  { //free in reverse order
              // of allocation
    while (x)
        if(fchar[fmark[--x]]->optr)
            free(fchar[fmark[x]]->optr);
}
```

```
pagebreak() { //page eject
   int i;
   char err;
   err = cvers(pldev,020);
   if ( err == -1 ) {
   printf("invalid filedes in pagebreak\n");
   exit();
   }
   for (i=0;i<TOP;i++) write(pldev,zero,2);
   linecount = TOP;
}

getdef()    {
   int blkc,bytc; register i;
   blkc = (hdr[infont*2]&0177400) >> 8;
   blkc =& 0377;
   bytc = hdr[infont*2+1];
   if(blkc)    {
      seek(fp,blkc,3); seek(fp,bytc,1); }
   else seek(fp,bytc,0);
   getnode();
   a->cc = infont;
   read(fp,&a->rw,2);
   read(fp,&a->lk,2); read(fp,&a->rft,2);
   read(fp,&a->drc,2);
   a->bytes = (a->rw%8 == 0) ? a->rw/8 : a->rw/8+1;
   if(a->drc)    {
      if((i=a->optr=alloc(a->drc*a->bytes)) < 0) {
         dealloc(nodeptr-1);
          getdef(); return;
      }
      read(fp,a->optr,a->drc*a->bytes);
   }
   in = 0;
   for(i=0;i<nodeptr;i++)    {
      if(fmark[i] == infont) in++;
   }
   if(in == 0) fmark[nodeptr-1] = infont;
}

getnode()    {
   if(nodeptr > 127)    {
      printf("overflow"); exit();}
   a = fchar[infont] = &clist[nodeptr++];
   a->optr = 0;
}

int roof(x)
   int x;    {
   x = (x%8 == 0) ? x/8 : x/8 + 1;
   if(x%2 == 0) return(x);
   if(x == 263) return(262);
   *++pl = 0; return(++x);
}
```

```
int check()    {
   if(ht < 0 || maxw < 0 || lht < 0 ||
      ht > 120 || maxw > 256 || lht > ht) return(1);
   else return(0);
}

int nroom(x)
   int x;    {
   if(linecount + x > PAGEHT) return(1);
   else return(0);
}

shift()    {
   int tb;
   tb = *lp; tb =& 0377; tb =<< openbits;
   if(r > 7)    {
      *pl++ =| (tb & 0177400) >> 8;
      *pl =& 0; *pl =| tb & 0377;
   } else    {
      if(r <= openbits)    {
         *pl =| (tb & 0177400) >> 8;
         openbits =- r;
      } else    {
         *pl++ =| (tb & 0177400) >> 8;
         *pl =& 0; *pl =| tb & 0377;
         openbits = 8-(r-openbits);
      }
   }
   lp++;
}

int oct(cp)
   char *cp;    {
   int i; i = 0;
   base = (*cp == '0') ? 8 : 10;
   while (num(*cp) && *cp != '\0')
      i = i*base + *cp++ - '0';
   return(i);
}

int num(cp)
   char cp;    {
   if(base == 10 && (cp >= '0' && cp <= '9')) return(1);
   if(base == 8 && (cp >= '0' && cp <= '7')) return(1);
   if (cp == '8' || cp == '9')   {
     printf("input error-- ");
     printf("\timproper octal number...%d",cp);
     while (*t != '\n') putchar(*t++);
     exit();
     }
   else return(0);
}

getch() {
   char tt,s;
```

```
    s = read(ip,&tt,1);
    if ( s == 0 ) return('\0');
    else return(tt);
}

flushh() { //print bad input line and exit
    while (*t != '\n') outchar(*t++);
    exit();
    }
```

BIBLIOGRAPHY

1. Barnett, Michael P., COMPUTER TYPESETTING, The M.I.T. Press, 1965.

2. Berg, N. Edward, ELECTRONIC COMPOSİTION, Graphic Arts Technical Foundation, 1975.

3. BOOK PRODUCTION INDUSTRY, "Morrow Issues First Book Published Entirely by MACHINE", March 1970, 46:55.

4. Boshak, V., "Nonimpact Printers", DATAMATION, May 1973, p 71.

5. Doyle, P.M., AN ADAPTATION OF THE HERSHEY DIGITIZED CHARACTER SET FOR USE IN COMPUTER GRAPHICS AND TYPESETTING, M.S. Thesis, Naval Postgraduate School, June 1977.

6. Hattery, Lowell H. and Bush, George P., eds., AUTOMATION AND ELECTRONICS IN PUBLISHING, Spartan Books, 1965, pp 9-17.

7. Naval Postgraduate School Technical Note NPS52Ba77061, A USER'S GUIDE FOR FONT CREATION AND MANIPULATION AT THE NAVAL POSTGRADUATE SCHOOL, by G.L. Barksdale, Jr., P.M. Doyle, and B.S. McCord, June 1977.

8. Ossanna, Joseph F., THE NROFF USER'S MANUAL, Bell Telephone Laboratories, Incorporated, 1974.

9. Ossanna, Joseph F., THE TROFF USER'S MANUAL, Bell Telephone Laboratories, Incorporated, 1974.

10. Ritchie, D.M., C REFERENCE MANUAL, Bell Laboratories, Incorporated, 1974.

11. Ritchie, D.M. and Thompson, K., THE UNIX TIME SHAPING SYSTEM, Bell Telephone Laboratories, Incorporated, 1974.

12. Stanford Artificial Intelligence Laboratory Operating Note 74, by Les Earnest, May 1976.

13. Thompson, K. and Ritchie, D.M., THE UNIX PROGRAMMER'S MANUAL, 6th ed., Bell Telephone Laboratories, Incorporated, Chapter I, 1975.

14. U.S. Department of Commerce NBS Monograph 99, AUTOMATIC TYPOGRAPHIC-QUALITY TYPESETTING TECHNIQUES: A STATE-OF-THE-ART REVIEW, by J.L. Little and M.E. Stevens, April 7, 1967, pp.

15. Usher, A.P., HISTORY OF MECHANICAL INVENTIONS, Beacon Press, 1959, p 6.

16. Yasaki, E.,"The Computer & Newsprint", DATAMATION, March 1963, p 27.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexanderia, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Professor Gerald L. Barksdale,Jr., Code 52Bk Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LCDR Stephen T. Holl, USN, Code 52Hl Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Captain B. Scott McCord, USMC 96 Orrlawn Drive Tallmadge, Ohio 44278	1